

CRANFIELD UNIVERSITY

MANUEL ANGEL AMARO CARMONA

4DT GENERATOR AND GUIDANCE SYSTEM

SCHOOL OF AEROSPACE, TRANSPORT AND  
MANUFACTURING

MSc-by-Research  
Academic Year: 2014-2015

Supervisor: Dr. Huamin Jia  
October, 2015



CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND  
MANUFACTURING

MSc by Research

Academic Year 2014-2015

MANUEL ANGEL AMARO CARMONA

4DT GENERATOR AND GUIDANCE SYSTEM

Supervisor: Dr. Huamin Jia  
October, 2015

This thesis is submitted in partial fulfilment of the requirements for  
the degree of Master of Science by Research

© Cranfield University 2015. All rights reserved. No part of this  
publication may be reproduced without the written permission of the  
copyright owner.



## **ABSTRACT**

This thesis describes a 4D Trajectories Generator and Guidance system. 4D trajectory is a concept that will improve the capacity, efficiency and safety of airspace. First a 4D trajectories synthesizer design is proposed. A flight plan composed by a set of waypoints, aircraft dynamics model and a set of limits and constraints are assembled into an optimal control problem. Optimal solution is found by making use of an optimal control solver which uses pseudo spectral parametrization together with a generic nonlinear programming solver.

A 4D Trajectories generator is implemented as a stand-alone application and combined with a graphic user interface to give rise to 4D Trajectories Research Software (4DT RS) capable to generate, compare and test optimal trajectories.

A basic Tracking & Guidance system with proportional navigation concept is developed. The system is implemented as a complementary module for the 4D trajectories research software.

Simulation tests have been carried out to demonstrate the functionalities and capabilities of the 4DT RS software and guidance system.

Tests cases are based on fuel and time optimization on a high-traffic commercial route. A standard departure procedure is optimized in order to reduce the noise perceived by village's population situated near airport. The tracking & guidance module is tested with a commercial flight simulator for demonstrating the performance of the optimal trajectories generated by the 4DT RS software.

Keywords:

Four dimensional navigation, optimal trajectories, optimization, guidance.



## CONTRIBUTION

This thesis makes a contribution to the scientific community in:

- Design and develop a 4D Trajectories Research Software to generate, evaluate and compare optimal trajectories.
- Design and develop a Tracking and Guidance System to test and evaluate optimal trajectories with a commercial flight simulator.

Furthermore, the author contributed to design, develop, integrate and test components of Greener Aircraft Trajectories under ATM Constraints (GATAC) framework developed by Airbus Group Innovations, Thales Avionics, German Aerospace Center (DLR), Alenia and GSAF group (*Cranfield University*, University of Malta, Technical University of Delft and Netherlands Aerospace Centre) as part of activities carried out by Systems for Green Operations (SGO) integrated technology demonstrator (ITD) of CleanSky programme.

Specifically, the author contributed to:

- Design, develop, integrate and test internal components (Weather and Atmospheric Models) of Aircraft Dynamics Model (ADM)
- Test and evaluate Business and Operational Cost Models (B&O)
- Test and evaluate GATAC user interface

These tasks were carried out as part of the 80-hours/month work basis required by CleanSky | Cranfield University.

# TABLE OF CONTENTS

ABSTRACT .....	i
CONTRIBUTION .....	ii
LIST OF FIGURES .....	vii
LIST OF TABLES .....	x
LIST OF EQUATIONS .....	xii
LIST OF ABBREVIATIONS .....	xiv
Chapter 1 .....	1
1.1 Motivation .....	1
1.2 Research Methodology .....	5
1.3 Research Objectives .....	7
1.4 Report Outline .....	8
Chapter 2 .....	9
2.1 Introduction .....	9
2.2 Flight Management Systems (FMS) .....	9
2.2.1 Boeing FMS .....	10
2.2.2 Airbus FMS .....	12
2.3 Future Flight Management Systems (FMS) .....	14
2.4 ARINC 424 Specification .....	16
2.5 Brief Review of Existent Tools .....	17
2.5.1 Center-TRACON Automation System (CTAS) .....	17
2.5.2 Greener Aircraft Trajectories under ATM constraints (GATAC) .....	19
2.6 4D Optimal Trajectories .....	21
2.6.1 4D Trajectory Generation for Waypoints-based Navigation .....	22
2.6.2 4D Trajectory-based concept for Terminal Area Operations .....	23
2.6.3 Optimal trajectories generation for next generation FMS .....	26
2.6.4 4D trajectory design in presence of contrails .....	27
2.7 Tracking & Guidance .....	29
2.7.1 4D Green Guidance for Terminal Area Operations .....	29
2.7.2 4D Tracking System for Waypoint-Based Navigation .....	31
2.7.3 4D Descent Trajectory Guidance .....	32
2.8 Optimal Control Methods .....	35
2.9 Conclusion .....	36
Chapter 3 .....	38
3.1 Introduction .....	38
3.2 Optimal Control Problem .....	38
3.3 Direct Methods .....	39
3.3.1 Direct Collocation .....	41
3.4 Tools for Solving Optimal Control Problems .....	43
3.5 Guidance using Proportional Navigation (PN) .....	46
3.6 Other Concepts .....	48

3.6.1 Andrew's Monotone Chain Convex Hull Algorithm .....	48
Chapter 4.....	50
4.1 Introduction .....	50
4.2 Aircraft Dynamics Model .....	50
4.3 Performance index specification .....	52
4.3.1 Reducing Fuel and Time .....	52
4.3.2 Reducing Noise .....	53
4.4 Boundaries and Constraints.....	57
4.5 Overview of 4D Trajectories Research Software (4DT RS) .....	60
4.6 Software Structure .....	62
4.7 4D Trajectories Generator Core.....	63
4.7.1 Exporting Results and Noise Grid .....	70
4.8 Graphic User Interface (GUI) .....	73
4.8.1 Maps, Waypoints and Data validation .....	73
4.8.2 Flight Plan Inspector.....	79
4.8.3 Aircraft and Initial Conditions Inspectors .....	83
4.8.4 Noise Inspector .....	83
4.9 4DT RS Core and 4DT GUI Connection .....	87
4.10 Computing Flight Information .....	89
4.10.1 Fuel, flight cost, distance and maximum SEL.....	90
4.10.2 Predicted Time of Arrival .....	91
4.10.3 Top of Climb and Begin of Decent .....	93
4.11 Trajectory Representation.....	96
4.11.1 Detailed View Tool .....	98
4.12 Preferences and Help .....	100
Chapter 5.....	102
5.1 Introduction .....	102
5.2 System Overview .....	102
5.3 Tracking System .....	105
5.4 Guidance System.....	107
5.4.1 Lateral Guidance .....	107
5.4.2 Vertical Guidance .....	111
5.5 Avionics Systems Indicators .....	112
5.6 MATLAB® Simulation Framework .....	113
5.7 4DT RS Guidance Module .....	114
5.7.1 Guidance System Core .....	114
5.7.2 4DT RS Guidance Graphic User Interface (GUI) .....	115
5.7.3 X-Plane® UDP Communication .....	117
Chapter 6.....	120
6.1 Introduction .....	120
6.2 Test case (Fuel and Time Optimization) .....	121
6.2.1 Setting Up Testing Case in 4DT RS .....	122



6.3 Test case (Noise Optimization) .....	128
6.3.1 Setting Up Testing Case in 4DT RS .....	131
6.4 Test case (Trajectory evaluation using 4DT RS T&G Module) .....	135
6.5 Known Limits and Improvements .....	140
Chapter 7 .....	142
7.1 Conclusion .....	142
7.2 Future Work .....	144
7.3 Challenges .....	145
REFERENCES .....	147
APPENDICES .....	155
Appendix A 4DT RS Overview .....	155
Appendix B Tracking & Guidance System Overview .....	156
Appendix C 4DT RS Functions .....	157
Appendix D MATLAB® Simulation Framework .....	163
Appendix E Noise Approximation of PW2036 engine .....	167
Appendix F Noise Approximation of PW2036 engine	<b>Error! Bookmark not defined.</b>



# ***LIST OF FIGURES***

Figure 1-1: Project methodology .....	6
Figure 2-1: Tools provided by CTAS suite [61].....	19
Figure 2-2: GATAC internal structure [32] .....	20
Figure 2-3: GATAC v3 typical display format of an optimization case. ....	21
Figure 2-4: Schematic of 4D Trajectory-based concept [45] .....	24
Figure 2-5: Schematic of solution proposed by [55] .....	28
Figure 2-6: Control Loop proposed by Vaddi et al. [44]. ....	30
Figure 2-7: Guidance and Control System Proposed by Canino et al. [21] .....	32
Figure 2-8: Overview of system proposed by Canino et al. [21] .....	34
Figure 3-1: Missile-target engagement geometry [54] .....	47
Figure 3-2: Monotone Chain   Convex Angles.....	49
Figure 3-3: Convex polygon example.....	49
Figure 4-1: Noise specification empirical data example .....	54
Figure 4-2: PW2036-like noise data approximation.....	56
Figure 4-3: Aircraft settings example.....	59
Figure 4-4: 4DT RS Graphic User Interface .....	61
Figure 4-5: 4D Trajectories Research Software Structure.....	63
Figure 4-6: 4DT RS Core internal structure.....	64
Figure 4-7: MappingWGS function algorithm.....	65
Figure 4-8: Compute_noise function algorithm.....	68
Figure 4-9: Integrand_cost function algorithm .....	69
Figure 4-10: Example of data output by 4DT RS Core .....	70
Figure 4-11: Noise Grid .....	71
Figure 4-12: Noise Grid Computation algorithm .....	72
Figure 4-13: PlotWaypoints function algorithm.....	75

Figure 4-14: PlotTrajectory function algorithm.....	76
Figure 4-15: Map and Flight Plan Table .....	78
Figure 4-16: Flight Plan Inspector .....	79
Figure 4-17: Navigation Database.....	82
Figure 4-18: Aircraft and Initial Conditions Inspector.....	83
Figure 4-19: Noise Inspector .....	84
Figure 4-20: Noise Specification File .....	85
Figure 4-21: FilterNoiseGrid function algorithm .....	87
Figure 4-22: 4DT RS Core and 4DT GUI Connection .....	88
Figure 4-23: Flight Information .....	89
Figure 4-24: ComputeFlightData function algorithm .....	91
Figure 4-25: Compute Time of Arrival Algorithm .....	93
Figure 4-26: Predicted Time of Arrival .....	93
Figure 4-27: Top of Climb (TOC) and Begin of Descent (BOD).....	94
Figure 4-28: <i>ComputeBODTOCPoints</i> function algorithm (Part 1) .....	95
Figure 4-29: <i>ComputeBODTOCPoints</i> function algorithm (Part 2) .....	96
Figure 4-30: <i>PlotGraph</i> function example in pseudo code and C# .....	98
Figure 4-31: FindNearestPoint function algorithm .....	99
Figure 4-32: Detailed View Tool .....	100
Figure 4-33: Preferences and Help .....	101
Figure 5-1: Tracking & Guidance System Main Control Loop.....	104
Figure 5-2: Compute track error loop .....	106
Figure 5-3: Lateral guidance based on static reference point.....	108
Figure 5-4: Compute reference point algorithm .....	110
Figure 5-5: Cross-track and altitude error indicators in PFD .....	112
Figure 5-6: Top-level view of tracking & guidance system .....	115
Figure 5-7: Tracking & Guidance System GUI .....	115
Figure 5-8: Example of reference point selection .....	116
Figure 6-1: Fuel vs Time Optimization   Map View .....	123

Figure 6-2: Fuel vs Time Optimization   Horizontal Profile.....	123
Figure 6-3: Fuel vs Time Optimization   Vertical Profile.....	124
Figure 6-4: Fuel vs Time Optimization   Speed Profile .....	126
Figure 6-5: Fuel vs Time Optimization   Fuel Consumption.....	126
Figure 6-6: Optimal trajectory compared to SID baseline procedures .....	131
Figure 6-7: Noise Exposure Level at Observer 1 (top) and Observer 2 (bottom) .....	132
Figure 6-8: Altitude Profile – Noise Optimization.....	133
Figure 6-9: Speed Profile – Noise Optimization.....	134
Figure 6-10: Horizontal Profile   Tracking & Guidance Module.....	137
Figure 6-11: Cross-track error for lateral guidance.....	137
Figure 6-12: Aircraft heading comparison .....	138
Figure 6-13: Vertical Profile   Tracking & Guidance Module.....	139

# LIST OF TABLES

Table 2-1: Boeing   Honeywell FMS [39] .....	11
Table 2-2: Boeing   General Electric (GE) FMS [39] .....	11
Table 2-3: Airbus   Thales FMS [39] .....	12
Table 2-4: Airbus   Honeywell FMS [39] .....	12
Table 2-5: RTA tolerance comparison [19] .....	15
Table 2-6: Performance Index Specification [45] .....	25
Table 3-1: Comparison between Optimal Control Tools .....	45
Table 4-1: Comparison between real noise data and approximated data .....	56
Table 4-2: Control variable constraints .....	58
Table 4-3: States variables constraints .....	58
Table 4-4: Default constraints .....	58
Table 4-5: <i>flat_point</i> struct .....	66
Table 4-6: PSOPT boundaries variable syntaxes in C++ .....	66
Table 4-7: <i>noise_data</i> struct .....	67
Table 4-8: Example of PSOPT setting variables sintaxis in C++ .....	70
Table 4-9: <i>PointLatLng</i> struct .....	73
Table 4-10: <i>GMapMarker</i> struct .....	73
Table 4-11: <i>GMapRoute</i> struct .....	74
<b>Table 4-12: Objects and elements used .....</b>	<b>74</b>
Table 4-13: WindowsForms Controls .....	77
Table 4-14: Validation functions .....	77
Table 4-15: Database specification .....	80
Table 4-16: Functions used by <i>SQLiteConnection</i> and <i>cmd</i> objects .....	80
Table 4-17: Variable values for SQL queries .....	81
Table 4-18: Noise Level Contours example values .....	85

Table 4-19: Convex Hull main functions.....	86
Table 4-20: Settings file parameters.....	88
Table 4-21: <i>FlightData</i> struct.....	90
Table 4-22: ZedGraph objects.....	97
Table 5-1: <i>UPD_Pack</i> specification [74].....	118
Table 6-1: LFPG – EHAM Flight Plan.....	121
Table 6-2: Initial Conditions.....	122
Table 6-3: TOC, BOD and PTA   Time Optimization .....	124
Table 6-4: TOC, BOD and PTA   Fuel Optimization .....	125
Table 6-5: Flight Information   Time and Fuel Optimization.....	127
Table 6-6: Enquiries registered in 2010 [35].....	129
Table 6-7: Visibility and Cloud configuration.....	136
Table 6-8: Wind speed configuration.....	136

***LIST OF EQUATIONS***

(3-1)..... 39

(3-2)..... 39

(3-3)..... 40

(3-4)..... 41

(3-5)..... 41

(3-6)..... 42

(3-7)..... 42

(3-8)..... 42

(3-9)..... 42

(3-10)..... 42

(4-1)..... 50

(4-2)..... 51

(4-3)..... 51

(4-4)..... 51

(4-5)..... 51

(4-6)..... 51

(4-7)..... 51

(4-8)..... 51

(4-9)..... 51

(4-10)..... 52

(4-11)..... 52

(4-12)..... 53

(4-13)..... 54

(4-14)..... 55

(4-15)..... 55



(4-16).....	55
(4-17).....	55
(4-18).....	55
(4-19).....	57
(4-20).....	60
(4-21).....	60
(4-22).....	71
(4-23).....	71
(4-24).....	92
(4-25).....	92
(5-1).....	105
(5-2).....	107
(5-3).....	108
(5-4).....	109
(5-5).....	110
(5-6).....	111
(5-7).....	113
(5-8).....	113
(5-9).....	113

# ***LIST OF ABBREVIATIONS***

3DoF	Three Degrees of Freedom
4DT	Four Dimensional Trajectories
4DT RS	4D Trajectories Research Software
6DoF	Six Degrees of Freedom
AI	Attitude Indicator
ARQ	Automatic Repeat Request
ATC	Air Traffic Controller
ATM	Air Traffic Management
BADA	Base of Aircraft Data
BIG	BIGGIN Waypoint
BOD	Begin of Descent
CLN	CLACTON Waypoint
CTA	Controlled Time of Arrival
CTAS	Control-TRACON Automation System
D2	Direct-to
DC	Douglas Company
DCNLP	Direct Collocation with Nonlinear Programming
DVR	DOVER Waypoint
DYNOPT	Dynamic Optimizer
EGKK	London Gatwick Airport
EHAM	Amsterdam Schiphol Airport
ETA	Estimated Time of Arrival
FAA	Federal Aviation Administration
FEC	Forward Error Coding
FMS	Flight Management System
FMC	Flight Management Computer
GATAC	Greener Aircraft Trajectories under ATM Constraints
GA	Genetics Algorithms
GE	General Electric
GNSS	Global Navigation Satellite System

GUI	Graphic User Interface
HYOP	Hybrid Optimizer
IFR	Instrumental Flight Rules
ILS	Instrument Landing System
INM	Integrated Noise Model
IPOPT	Internal Point Optimizer
LFPG	Paris Charles de Guille Airport
MACS	Multi-Aircraft Control System
MINLP	Mixed Integer Nonlinear Programming
MOTS	Multi-objective Tabu Search
NAS	National Air Space (United States)
NASA	National Aeronautics and Space Association
ND	Navigation Display
NDB	Non-directional beacon
NDB	Navigation Database
NG-FMS	Next Generation Flight Management System
NLP	Nonlinear Programming
NSGAMO	Non-dominated Sorting Genetic Algorithm Multiple Optimization
OCP	Optimal Control Problem
PFD	Primary Flight Display
PMM	Point Mass Model
PSOPT	Pseudo Spectral Optimal Control Solver
RNP	Required Navigation Performance
RTA	Required Time of Arrival
SEL	Sound Exposure Level
SESAR	Single European Sky ATM Research
SGO	Systems for Green Operations
SID	Instrument Standard Departure
SNOPT	Sparse Nonlinear Optimizer
STAR	Standard Arrival
T&G	Tracking & Guidance
TOC	Top of Climb
VOR	VHF Omnidirectional Range



# ***Chapter 1***

## **INTRODUCTION**

This chapter describes a background about current systems that include four dimensional trajectories and the reasons that inspired to carry out this project. Then a formal thesis definition is presented, including the general and specific objectives of this research and the methodology to achieve these objectives. Finally a report outline provides the reader with an overview of chapters contained in this document.

### **1.1 Motivation**

Four dimensional trajectory based operations (4D-TBO) is considered a key improvement for future Air Traffic Management systems [38].

Strategies involving arrival sequences based on a time of arrival while maintaining optimum flight profiles contributes to increase the airspace capacity, efficiency and safety. The main purpose of using 4D trajectories is to increase the predictability of this environment.

Despite a few efforts done by industry, progress in the use of four dimensional trajectories in both air and ground system is limited.

Smiths Aerospace (GE Aviation), is responsible for manufacturing the most recent flight management systems for Boeing B737-NG (-700, -800 and -900). Smith has implemented initial four dimensional capabilities in their FMS GE U10.7, by including a Required Time of Arrival (RTA) function [27].

This function operates based on predicted trajectory while associating the required time of arrival (input by crew) with the estimated time of arrival (ETA). Subsequently, aircraft is guided to the target waypoint so the predicted time of arrival matches as much as possible with the required time of arrival.

Similar functions has been included in many other Flight Management Systems manufactured by Honeywell or Thales and are mentioned in [18], [24], [31], [34] and [39].

One of the problems related to the use of RTA functions in the current airspace is that not all aircraft are provided with flight management systems capable to meet required time of arrival conditions. This input a homogeneity issue to airspace, since aircraft that are not equipped with RTA functions have to be accommodated in airspace along with others that include these functions. A transition period is ahead, and RTA traffic has to be compatible with non-RTA traffic, at least for terminal based operations. A key feature to achieve this, is the capacity to accurate predict optimal trajectories so RTA flights will be easier to handle.

Current scenario shows that RTA function is only available in some aircraft. These functions try to minimize throttle activity by triggering control inputs if RTA and ETA differs by more than a configurable tolerance. Despite this tolerance varies from  $\pm 6$  to  $\pm 30$  seconds in existent flight management systems, results of flight tests including possible future scenarios [19] show that capabilities to predict optimal trajectories need to be improved.

Another problem is produced when using ATC speed constraints (e.g. 250 kts. below FL100). Current RTA functions have been limited to operate at flight levels above FL100. Late arrivals are produced because the RTA function estimates inaccurately the time required to decelerate to constraints speeds. In some cases, predicted trajectories by FMS do not include speeds below 250 knots, even at levels above FL100.

Estimated Time of Arrival (ETA) is not updated frequently in current flight management system when target speeds cannot be properly controlled. This

deficiency in ETA updating rate could induce to inaccuracies in computation of flight parameters to follow the RTA condition.

Predicted optimal trajectories by current FMS are no longer reliable when RTA specifications are input after Begin of Descent (BOD) point has passed. The descent strategy is fixed when this point is achieved.

Current RTA tolerances are sufficient for some flight phases such as en-route or ascent. However, reducing these tolerances will be necessary for arrival and approach phases thus terminal area operations require more accuracy in aircraft separation.

In current FMS-Control Display Units (CDU), RTA error is displayed numerically [34], which could represent an issue for pilots that must have their head looking down for a considerable amount of time. Improvements in avionics displays is another important issue that has to be considered in the implementation of future flight management systems. Additionally, cockpit improvements must be complemented with GPS time inputs because it is necessary some reliable synchronization between ground and air systems.

Another important problem is related to compatibility of avionics systems to ground ATM tools. In situations including initial four dimensional trajectories (i4DT) concept, where required time of arrival is involved at only few merging points, the specific RTA condition at a sequencing fix could be negotiated by voice communications. However, for full 4D trajectories operations, where several sequencing fix are required, this situation has to be implemented in different ways. For this case, prediction of 4D trajectories has to be performed in both air and ground systems.

From the aircraft point-of-view, it is necessary to calculate the best trajectory that fulfil its interest (e.g. predict optimal trajectory to reduce fuel consumption). In contrast, for ground systems, it is necessary to calculate the best trajectory that improves the air traffic flow. A solution for this observation would be implementing optimal trajectories that meet both requirements.

The use of existent ATM tools such as Arrival Manager (AMAN) [30] in Europe airspace is still under consideration. The tool supposes to provide support for controllers by generating aircraft descent speeds on continuous descent approach operations. However, its predictability and operational issues due to incompatibility with avionics RTA functions requires improvements. Optimal descent paths predicted by existent FMS systems are not compatible with AMAN speed advisories. Another similar tool that includes a four dimensional synthesizer, Control-TRACON Automation System (CTAS) [52] has been developed for research and simulation purposes with an insight of NextGen project requirements. However, possibly due to its undeveloped level, it has only been included in small scales on Air Traffic Management Systems for research and testing purposes.

Assuming the diversity of problems described, it is noticeable that design goals for future air and ground systems are to generate four dimensional optimal trajectories capable to reduce fuel consumption or flight time while maintaining predictable paths (horizontal and vertical) and improving air traffic flow efficiency.

Current systems show that predicting optimal trajectories is far from being perfectly implemented. Most of advanced four dimensional features in air and ground systems are still under consideration or have not been properly developed because of deficiencies in the estimation of four dimensional trajectories. This project contributes to improve the predictability of four dimensional optimal trajectories by designing and developing a four dimensional trajectories synthesizer.

Additionally, the idea of progressing in the development of optimal trajectories and flight profiles is considered important to environment conservation because it reduces CO<sub>x</sub> emissions and reduce noise. Also, it is considered attractive for airlines, avionics systems manufacturers and air traffic management operators thus it reduces aircraft fuel consumption, flight time and operational costs, while improving the efficiency and increasing the airspace capacity.

The issue of predicting optimal trajectories restricts the current FMS-RTA functions, to a point, where they are not reliable and accurate enough in some



situations. Additionally, advanced RTA functions need to be properly integrated into flight deck displays to complete appropriate interaction with pilots. For this reason, this project also contributes to improve current avionics systems by proposing possible solutions to some of these issues.

## **1.2 Research Methodology**

General requirements of the system have been analysed as part of first specific objective of this project. Subsequently, the process to carry out this research has been divided into two phases:

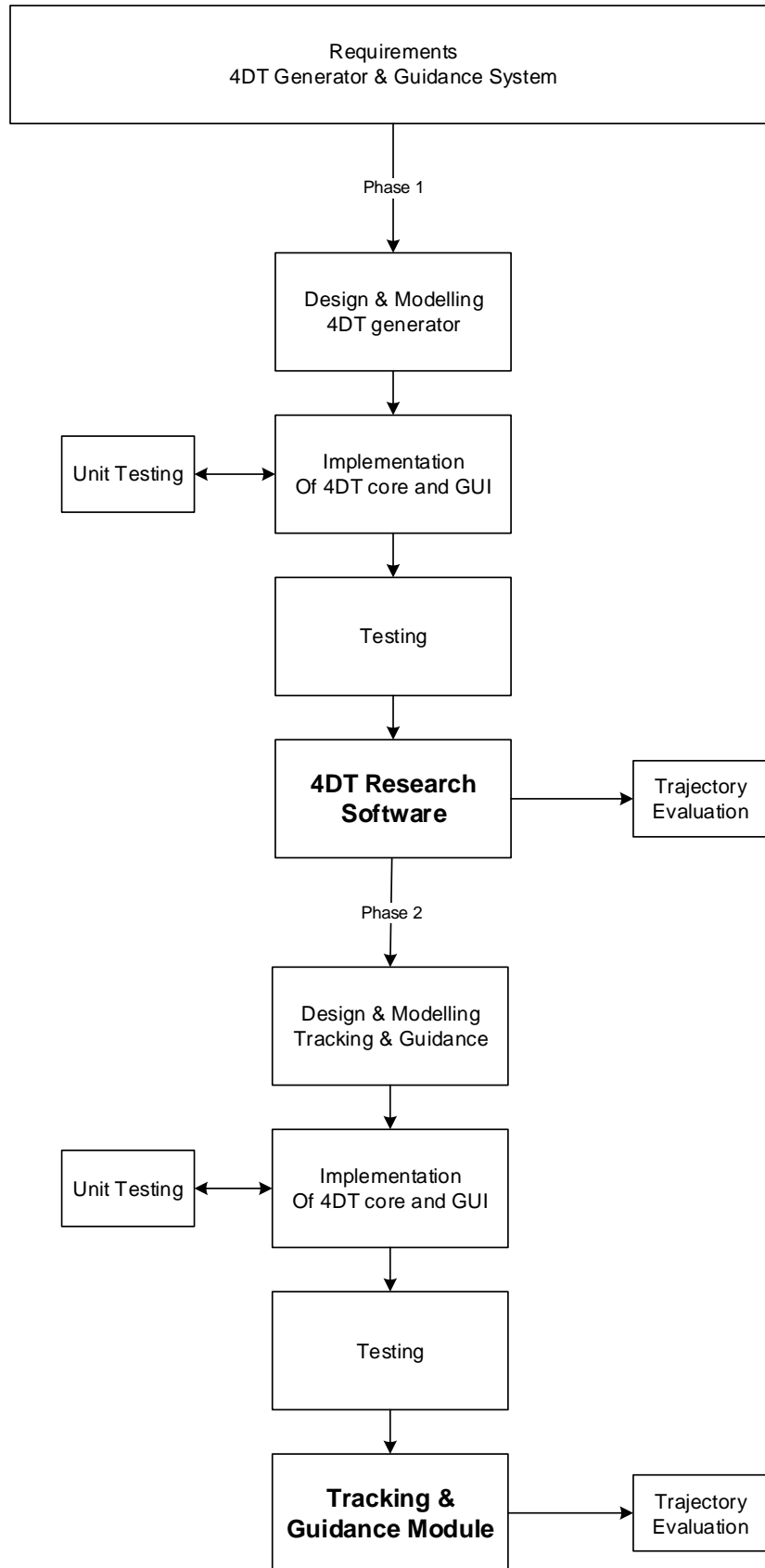
- *Phase 1: Design, Implementation & Testing of 4DT Generator*
- *Phase 2: Design, Implementation & Testing of Tracking & Guidance System*

Since project goals were evidently understood, the definition of the project was firm and most of the technologies needed were clearly understood, it has been used a sequential design process based on flexible and iterative Waterfall model [28].

Figure 1-1 shows the methodology used to develop this project including two phases following the waterfall design strategy.

Main phase boards the modelling, design and implementation of 4DT generator system. Initial tasks were mainly focused on generating optimal trajectories by setting up specific optimal control problems and identifying the general characteristics of these problems. These characteristics composed an important part of the system requirements.

Similarly, for the software design it has been considered also accessibility to tools and software constraints such as solver selection, operating systems and computer resources.



**Figure 1-1: Project methodology**

System implementation has been started when modelling & design stage was completed. Iterative procedures between these two stages (implementation ↔ design), have been performed to include elements that were not considered in early design stages.

Early unit testing have been performed since first versions of the synthesizer (4DT core) were released. This phase has been concluded with the test of alpha versions of 4DT research software which included most recent version of 4DT generator.

The second phase has been focused on design, test and implementation of the tracking & guidance system.

Design methodology used for this phase was similar to main phase. Waterfall strategy has been adapted to available time for the project. In this case, implementation stage included massive unit testing due to configuration and testing of commercial flight simulator.

At the end of second phase, the Tracking & Guidance system has been tested before evaluating any interaction with research software.

The completion of second phase conducted the testing and evaluation of 4DT synthesizer results (*Trajectories Evaluation*) and the end of the project.

### **1.3 Research Objectives**

This thesis aims at designing, modelling and testing a trajectory generator and guidance system to focus on the use of 4D optimal trajectories.

In order to achieve this, it has been proposed the following specific objectives:

- Analyse the functionalities of existing Flight Management Systems (FMS).
- Study functional requirements for future 4DT/RNP-based FMS.
- Design and develop a future 4DT Generator and Guidance system for a FMS according to the functional requirements analysed.
- Analyse and evaluate the performance of the 4DT Generator and Guidance system.

## 1.4 Report Outline

This report is composed by the following chapters:

**Introduction:** this chapter provides a general overview of this master thesis by explaining the motivation and objectives set.

**Literature Study:** this chapter provides an overview of current 4DT systems as well as optimal trajectories prediction and guidance system researches.

**4D Trajectory Optimization Techniques:** concepts related to optimal trajectory prediction, solving methods for optimal control problems, non-linear programming techniques and guidance based on proportional navigation are detailed and explained in this chapter.

**4D Generator Development:** this chapter provides detailed information about the design and implementation of 4DT Generator and 4D Trajectories Research Software v0.1.

**Tracking & Guidance System:** this chapter provides detailed information about design and implementation of Tracking & Guidance system.

**Test and Results:** this chapter describes 4DT Trajectories Research Software and Guidance system tests and results. Also provides an extensive analysis about capabilities and proposed improvements for designed systems.

**Conclusion and Future Work:** in this chapter it is summarized the work performed and it is contrasted with the project objectives. Finally it is given an insight of future work.

# ***Chapter 2***

## **LITERATURE STUDY**

### **2.1 Introduction**

This chapter will briefly review current and future Flight Management Systems (FMS) practices and requirements, 4D trajectory generation and guidance methods as well as trajectory optimization methods. Section 2.2 analyse the current Flight Management Systems (FMS) designs, especially FMSs equipage on Boeing and Airbus aircraft. Section 2.3 reviews future flight management systems, their functions and requirements. Section 2.4 describe navigation and database specifications used on current FMSs. Section 2.5 reviews existent tools to generate four dimensional trajectories. Section 2.6 discuss existent strategies and methods to generate optimal trajectories. Section 2.7 discuss existent tracking and guidance systems. Section 2.8 describes existent optimal control methods used for generating four dimensional trajectories. Finally, Section **Error! Reference source not found.** summarizes this chapter and identify the functional requirements of this project.

### **2.2 Flight Management Systems (FMS)**

First integrated Flight Management Systems (FMS) were introduced in late 1970s. Since then, it has been introduced more advanced functions and capabilities such as improvements in navigation database, enhanced tracking for vertical and horizontal paths, and basic required time of arrival functions. This section aims at describing current FMS in the market and their functions.

### 2.2.1 Boeing FMS

First Flight Management System (FMS) was developed for Boeing 767 programme in 1978. It included a system capable to automate a variety of in-flight tasks. In 1990, Boeing Company launched the B767-300 that included a Honeywell Pegasus 2005 Flight Management System [39].

The airline industry required new capabilities such as the *direct-to* function that describes aircraft trajectories without the need to follow nav aids. For this purpose, it has been implemented algorithms based on multiple sensors that provide more accurate aircraft position by making use of VHF Omni-directional range (VOR), Distance Measurement Equipment (DME) and Inertial Reference Systems (IRS).

Additionally, the increase in the demands on oceanic operations encouraged the development of Future Air Navigation Systems (FANS) capabilities that include the use of Global Position System (GPS), Required Navigation performance (RNP) and data link. This supposed an important advance on airline operational communications, more safe and reliable systems capable to track horizontal and vertical paths with very low tolerance errors defined by RNP conditions.

Except Boeing B737 versions, most airplanes of the American company use Flight Management Systems (FMS) manufactured by Honeywell International, Inc. Some famous models included by Boeing are Honeywell Legacy, AIMS, Pegasus and NextGen.

Honeywell develops a wide variety of systems that includes differences in hardware and software. Hardware is commonly compared by its capacity to host a Navigation Database (NDB).

FMS developed for B757/B767 (Honeywell 200K FMC) were provided with just 400KB of capacity. The introduction of Performance Base Navigation (PBN) supposed an important increase in the demand of database capacity. Nowadays, modern FMS developed for B747-8 (Honeywell NextGen) is provided with 100MB [39].

The following Table 2-1 shows most common Honeywell systems used in Boeing aircrafts, their Navigation Database (NDB) capacity.

**Table 2-1:** Boeing | Honeywell FMS [39]

<b>Aircraft</b>	<b>FMS Model</b>	<b>NDB Capacity</b>
B747-400	Honeywell 747-4	2MB
B747-8	Honeywell NextGen	100MB
B757/767	Honeywell Pegasus	7.5MB
B777	AIMS	12MB
B787	Honeywell	30MB

Most recent B737 versions (B737-NG), are provided with Flight Management Systems developed by GE Aviation (Smiths Aerospace). Different models has been developed for B737, some of them including new FMC hardware are provided with up to 16MB of Navigation Database capacity.

Following table shows a comparison between different FMS manufactured by GE for Boeing B737-NG.

**Table 2-2:** Boeing | General Electric (GE) FMS [39]

<b>Aircraft</b>	<b>FMS Model</b>	<b>NDB Capacity</b>
B737-NG	GE U10.6, U10.7, U10.8	8MB
B737-NG	GE U10.8	16MB

## 2.2.2 Airbus FMS

Most Airbus airplane models are provided with Thales Flight Management Systems (FMS). Exception applies for some aircraft of A320 family and A330/340 thus could be equipped with Honeywell FMS [39].

Flight Management Systems 1 (FMS1) developed by Thales was introduced in the Airbus A320 programme. The Thales FMS1 was provided with 400Kb of NDB capacity. Next versions FMS2 REV2+ and FMS2 R1-A increased their NDB capacity up to 7MB [39].

In the versions of A320 including Flight Management Systems manufactured by Honeywell, it is possible to highlight the advanced-capacity Pegasus P1-A that provided 20MB of NDB capacity.

The following Table 2-3 and Table 2-4 show most common FMS used by Airbus aircraft. It is shown that system capacities are relatively similar to models shown in section 2.2.1.

**Table 2-3:** Airbus | Thales FMS [39]

<b>Aircraft</b>	<b>FMS Model</b>	<b>NDB Capacity</b>
A319/320/321	Thales FMS2 REV2+	5MB
A319/320/321	Thales FMS2 R1-A	7MB
A330/340	Thales FMS2 REV2+	7MB

**Table 2-4:** Airbus | Honeywell FMS [39]

<b>Aircraft</b>	<b>FMS Model</b>	<b>NDB Capacity</b>
A319/320/321	Honeywell Pegasus P1	4MB
A330/340	Honeywell Pegasus P3	5.5MB



Boeing and Airbus use very similar Flight Management Systems (FMS) where few variances could be detected and mostly related to the diversity of manufacturers. Important changes in functions and the NDB capacity can be perceived when comparing recent systems with first generation models.

Designing methods and techniques for most Flight Management Systems (FMS) are not in public domain, however it is known that these methods rely on managing Navigation Databases (NDB) that are defined via ARINC-424 specification [58].

From the user perspective, most differences between different Flight Management Systems are almost imperceptible. However, there exist few cosmetic changes in flight deck or software front-end that are visible.

Most changes are performed in the hardware, in the system implementation or in the software design. Previous discussion boarded differences in navigation database capacities. Furthermore, since manufacturers are different, it could exist dissimilarities in the way trajectories are computed or in the perception of some aircraft parameters.

A research carried out by Herndon et al. [24] from MITRE Corporation Center for Advanced Aviation Systems Development studied the differences between Flight Management System (FMS) and Flight Management Computers (FMC) designed by different manufacturers. The study results show that there exist differences below 0.1 nautical miles of the nominal radius-to-fix (RF) arc radius in the turn. In vertical path, all tracks were followed with errors below 200-300 feet. These results show that differences in lateral and vertical path exist, however the tracking deviations are within published maximum tolerances.

Another study carried out by same author [18], shows a difference between vertical paths of diverse Flight Management Systems. The results shown that climb paths could be more widely different than descent paths. While comparing different FMS included in Boeing and Airbus models, vertical path constraints were generally well met by all systems with exception of one system. However, the author states that some tests could be affected by non-related FMS effects.

As a general comparison, it is possible to affirm that difference between different manufacturers are not excessively significant, however each system is developed by different manufacturers that sometimes input variances in each design.

## **2.3 Future Flight Management Systems (FMS)**

Single European Sky ATM Research (SESAR) [38] programme and the United States NextGen [26] aim at restructuring the airspace to increase the overall capacity and efficiency by introducing more strategic-based operations instead of current tactical operations.

Both programmes require to develop and implement the use of 4D trajectory based operations (4D-TBO) in both air and ground systems in order to improve the predictability of the airspace.

The premise of SESAR and NextGen is based on sequencing the air traffic by providing each aircraft with a set of waypoints that aircraft have to reach at required time. The required time of arrival (RTA) should be communicated to crew by air traffic controller (could be negotiated).

From aircraft operator's point of view, the use of 4D-TBO will result economically interesting. Aircraft will be sequentially managed based on required time of arrival constraints that will result in a reduction of *bottle-neck* effects and holding patterns and consequently a considerable reduction of fuel consumption, hence CO<sub>x</sub> emissions.

In contrast, air traffic management systems will use 4D-TBO to increase the overall efficiency of traffic flow. Additionally, airspace capacity will be significantly increased due to reduction of *waiting* time in the air, holding patterns and airport delays.

For operations involving few merging points, the communication of RTA could be performed by voice commands. However, for operations involving more points, it is necessary to modify both air and ground systems to provide this interaction in more efficient manner. For this reason, future flight management systems require advanced data-link functions capable to interact with ground ATM systems. The

process should be easier to handle than current functions by introducing more efficient software interfaces.

Few advances has been performed in the use of four dimensional trajectories concept in avionics systems. The most significant function is named *Required Time of Arrival (RTA)* [23].

This function operates based on aircraft trajectory predictions to calculate the Estimated Time of Arrival (ETA). Once RTA function is activated, the aircraft speed up or down in order to maintain the difference between RTA and ETA below a tolerance value.

The following table shows a comparison of RTA tolerance between different FMS manufacturers:

**Table 2-5: RTA tolerance comparison [19]**

<b>FMS</b>	<b>Aircraft</b>	<b>RTA Tolerance</b>	<b>Flight Phase</b>
Smiths	B737 Classic, NG	6 sec	Climb, Cruise, Descent
Thales - Smiths	A320, A330, A340	30 sec	Climb, Cruise, Descent
Honeywell Pegasus	A320, A330, A340 B757, B767, MD90	30 sec	Cruise
Honeywell	B777, B747-400, MD11	30 sec	Cruise

Required Time of Arrival (RTA) functions exists in modern airliners but they have limitations and require special autopilot functions such as auto-throttle to operate.

RTA functions in current FMS do not operate for altitude below 10,000 ft. For this reason, future flight management systems should include RTA functions capable to operate in approach phases. Since reduced aircraft separation on terminal

area operations is a key requirement, lower tolerance errors for RTA functions should be achieved.

Prediction of optimal trajectory is an important feature of flight management systems. For the particular case of four dimensional trajectories, Estimated Time of Arrival (ETA) is used to compute required control inputs to match the required time of arrival. However, updating rates of estimated time of arrival is reduced when aircraft speed do not match properly with control speed which results in accuracies when reaching the waypoint. Future flight management systems should include important advances in trajectory prediction that support the control activity of RTA functions more accurately.

## **2.4 ARINC 424 Specification**

ARINC 424 is a specification that has been released for first time in May 1975 [59]. It is maintained and developed by the industry and it is used for exchange of communication and navigation data between avionics systems and data providers.

This specification was created specifically for avionics systems that require complex navigation and communication data. For this reason, ARINC-424 specification has been mainly developed thinking on its application on Flight Management Systems (FMS) and Flight Management Computers (FMC).

The main idea behind ARINC 424 is to specify data in text files containing lines of 132 alphanumeric characters [65]. This data is assembled and packed by commercial suppliers. These text files are converted into binary datasets and included into avionics systems by Flight Management Systems (FMS) manufacturers. In this way, each Flight Management System (FMS) is provided with a unique set of navigation and communication data.

ARINC 424 document provides a specification about how a navigation database should be assembled and prepared by commercial suppliers. In addition, it provides a set of rules to avionics manufacturers about how this data can be accessed and interpreted by embedded systems. ARINC 424 specifies standard assembly information for the following navigation elements: airports, heliports,

runways, waypoints, navaids (e.g. NDB, VOR, VOR-DME, TACAN, ILS), airways and arrival/departure routes procedures (STAR, SID).

World's largest navigation and communication database supplied by Jeppesen® meet ARINC 424 specification [58]. This database is currently used by several manufacturer to fly, simulate and create flight plan procedures.

Additionally ARINC 424 specifies a set of legs used for coding Terminal Area Procedures, Standard Instrument Departure (SID) and Standard Arrival (STAR) procedures. When this standard was introduced, it was required to implement RNAV procedures using more elements than the few of them specified in DO-236\* standard [14]. For this reason, ARINC-424 introduced a set of 23 new legs types in order to translate normal procedures created for compass and manual flight into computer language (for Flight Management Systems).

## **2.5 Brief Review of Existent Tools**

The introduction of SESAR and NextGen programmes supposed that 4D trajectory-based operation concept became more popular in recent years. Some tools including 4D optimization capabilities have been developed and are described in this section.

### **2.5.1 Center-TRACON Automation System (CTAS)**

The Aviation Systems Division of NASA Ames Research Center developed an important suite of tools for synthetizing optimal trajectories. Center-TRACON Automation System (CTAS) [52] is a set of tools that has been developed to complement ATM ground systems by including a next level of automation for planning and controlling arrival and departure traffic. The core of CTAS is composed by a *trajectory synthetizer* that generates trajectories to increase fuel efficiency, and provide automation assistance to air traffic controller in order to reduce traffic delays.

CTAS also complements its features with En-Route Descent Advisor (EDA), which is an internal tool that assists air traffic controllers of each airspace sector to solve air conflicts.

The core of this software is a sophisticated trajectory synthesizer that aims at predicting trajectories based on flight data information hosted in an ATC computer. This synthesizer makes use of accurate aircraft performance models located in this ATC database.

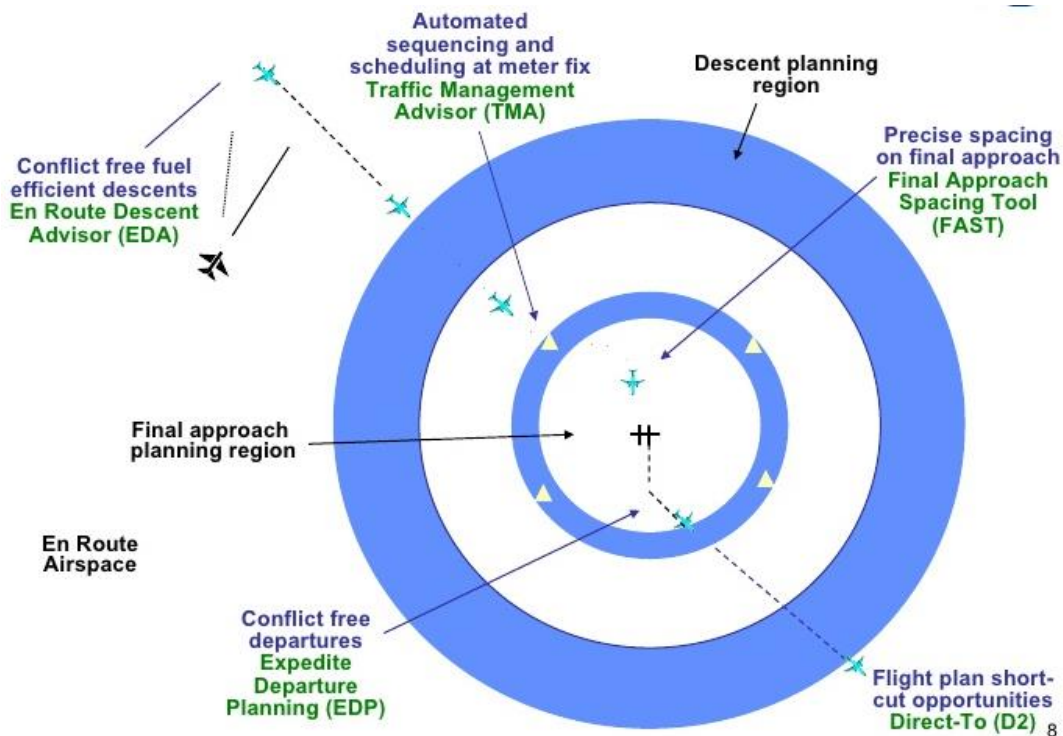
4D trajectories concept has been introduced in this tool in order to create schedules for runway occupancy and final approaches. Also, the introduction of time as new aircraft trajectory dimension allows to detect future conflicts, suggest possible shortcuts and create more efficient descent paths.

CTAS has been focused of making use of sophisticated 4D trajectories based functions. The En-Route Descent Advisor (EDA) and the Active-Final Approach Spacing Tool (A-FAST) allows to generate trajectories that meet time of arrival. According to [52], advisories are displayed by a module called Traffic Management Advisor (TMA) in order to issue guidance commands by air traffic controllers to avoid aircraft conflicts.

CTAS deals with operational challenges that result from predicting 4D optimal trajectories such as human factor problems in communication or monitoring, imprecision in execution of air traffic controllers commands by crew or automated systems and environment uncertainties that affect trajectory predictions.

From the optimization point-of-view, an interesting tool provided by CTAS is Direct-To (D2). In order to decrease fuel consumption and aircraft CO<sub>x</sub> emissions, D2 allows controllers to visualize and if required, to modify aircraft trajectories to make trajectories shortcuts available in the route. If this is the case, aircraft are capable to save time and in some cases, contributes to dynamically release the airspace capacity.

The capabilities of CTAS have been tested and transitioned to FAA for operational evaluation. Other modules of this suite are detailed in [52], [15] and [61]. In addition, Figure 2-1 shows a diagram of different tools provided in CTAS Suite in February, 2007 including their region of action.



**Figure 2-1:** Tools provided by CTAS suite [61]

### 2.5.2 Greener Aircraft Trajectories under ATM constraints (GATAC)

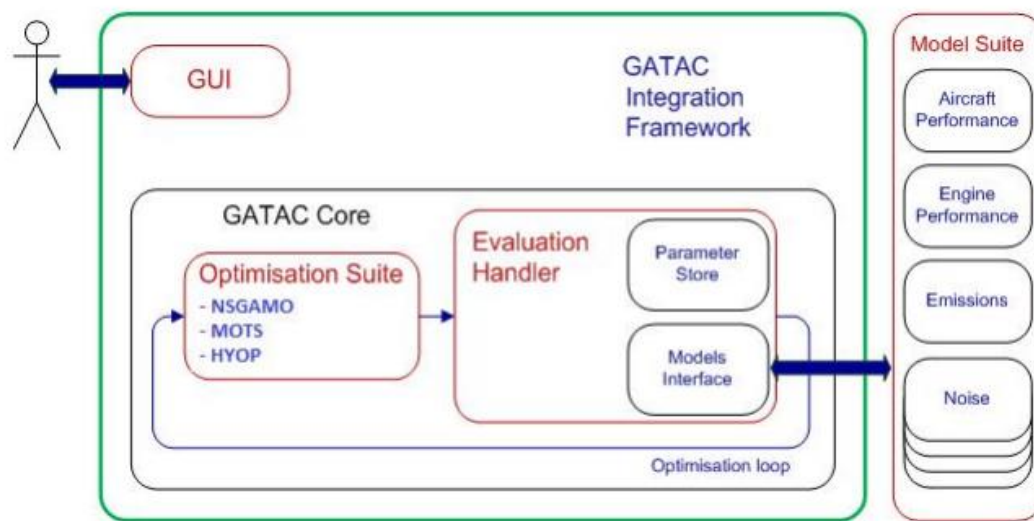
CleanSky is a joint technology initiative created by European Commission in 2008 in order to develop new technologies capable to significantly increase the environmental performances of air transport vehicles [60]. The innovative projects are integrated and coordinated by technology demonstrators. Cranfield University contributes in activities related to Systems for Green Operations (SGO) of CleanSky.

Greener Aircraft Trajectories under ATM constraints (GATAC) is a multi-model and multi-optimization framework developed by integrated technology demonstrators and associates of CleanSky Work Package WP3.2, which includes GSAF (Group composed by Cranfield University, University of Malta, NLR and TU Delft), Airbus Group Innovations, Alenia Aeronautica, DLR and Thales.

GATAC core is composed by an optimization suite which uses different optimization algorithms depending of user or problem requirements. These

algorithms are Non-dominated Sorting Genetic Algorithm Multiple Optimization (NSGAMO), Multi-objective Tabu Search (MOTS) and Hybrid Optimizer (HYOP) [42]

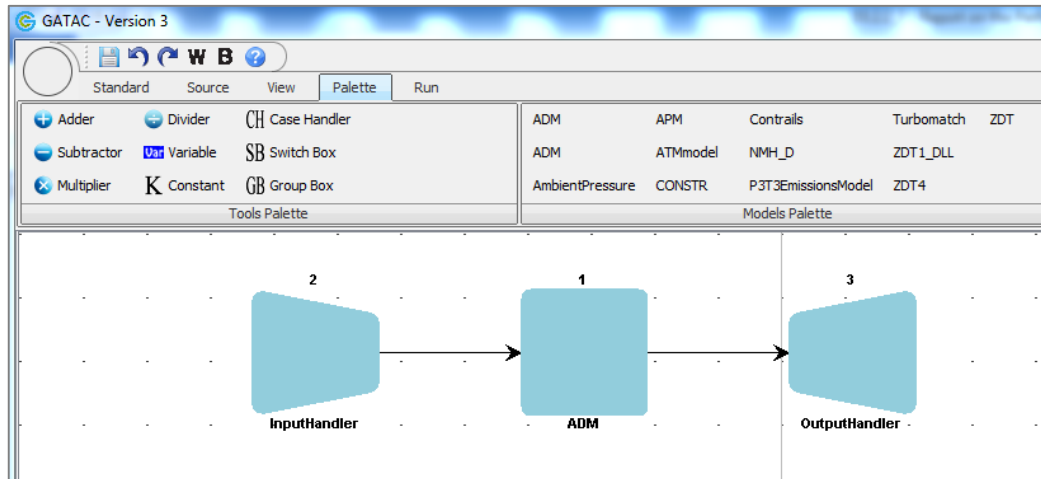
Figure 2-2 shows a structure of GATAC software. The optimization suite uses an evaluation handler which is composed by a parameter store component and an interface used to connect different models contained into a models suite.



**Figure 2-2:** GATAC internal structure [32]

From the user point-of-view, GATAC is a software compatible with Windows operating systems. The Graphic User Interface (GUI) allows users to manage the models and components of the optimization process represented by *blocks* that can be connected to each other.





**Figure 2-3:** GATAC v3 typical display format of an optimization case.

Multi-objective and multi-model optimization characteristics convert this framework into a powerful tool. However, due to the nature of genetics-based algorithms, thousands of iterations are needed in order to obtain a successful result. This implies the use of powerful CPUs and/or local area network clusters in order to speed the optimization process up. Another important characteristic of GATAC is that generated trajectories are exported to a data text file. Therefore, third-party software is required to plot generated trajectories.

Main focus of GATAC is generate optimal trajectories making use of accurate aircraft models that include many modules such as engine, aircraft dynamics and weather conditions. At this moment it has not shown evidence of providing support to operations involving 4D trajectories concept such as managing optimal trajectories making use of scheduling constraints or defining a predicted time of arrival relative to a merge point, however its flexibility could allow users to develop models that include this functionality in the future.

## 2.6 4D Optimal Trajectories

In general, optimal trajectories generation is usually boarded as an *optimal control problem* based on mathematical models that describes aircraft dynamics, a cost function describing an aircraft performance specification to be minimized (fuel consumption, flight time and others) and a set of equality and/or inequality constraints describing the aircraft performance limits. This section describes and

analyses different methods and approaches to generate 4D Optimal trajectories followed by other authors.

### 2.6.1 4D Trajectory Generation for Waypoints-based Navigation

Bouson and Machado [49] describe a method for generating 4D optimal trajectories passing through a sequence of waypoints using pseudo spectral optimization. The method described used simple navigation equations that provide a reference velocity ( $V_{ref}$ ), path angle  $\gamma$  and heading  $\psi$  to enable aircraft go through a sequence of waypoints denoted by  $P_0, P_1, P_2, P_3 \dots P_M$ .

The basic premise is to consider the time in the description of waypoints so that each point is defined by  $P_k = (\lambda_k, \varphi_k, h_k, t_k)$  where variables are longitude ( $\lambda$ ), latitude ( $\varphi$ ), altitude ( $h$ ) and time ( $t$ ). Then the problem is defined as an optimal control problem to be applied for points  $P_0, P_1, P_2, P_3 \dots P_M$  that minimizes the arrival delay at each point  $P_k$ .

Hence the following performance specification is proposed:

$$J(u) = \left( P_f - s(\tau_f) \right)^T Q_f (P_f - s(\tau_f))$$

where  $s(\tau_f)$  is the terminal position of aircraft,  $Q_f$  a positive defined matrix of appropriate dimension, and  $u$  is the control vector of the following navigation model [49]:

$$\dot{\lambda} = \frac{V \cos(\gamma) \sin(\psi)}{(R_e + h) \cos(\varphi)}$$

$$\dot{\varphi} = \frac{V \cos(\gamma) \sin(\psi)}{R_e + h}$$

$$\dot{h} = V \sin(\gamma)$$

$$\dot{V} = u_1$$

$$\dot{\gamma} = u_2$$

$$\dot{\psi} = u_3$$

Where  $\lambda$  is longitude,  $\varphi$  is latitude,  $h$  is altitude,  $V$  is the velocity,  $\gamma$  is the flight path angle and  $\psi$  is heading. The control variables selected are acceleration, flight path rate and heading rate.

The optimal control problem is then restricted by a set of boundaries and constraints. The constraint defined by the following expression:

$$\|P_k - s(\tau_k)\|_2 \leq \sigma$$

is used to ensure aircraft passes through all waypoints ( $k = 1 \dots m$ ) at time denoted by  $\tau_k$  with a tolerance position denoted by the variable  $\sigma$  which represents the radius of a circle with center fixed at the point  $P_k$ .

Finally, the trajectory is generated by discretizing the time dependent equations using a pseudo spectral method based on Lagrange and Chebyshev polynomials defined in the interval  $[-1, 1]$ . Other tests have been performed using another Collocation method. The nonlinear programming problem obtained is then solved using the function *fmincon* of MATLAB®.

The method used by the author and the reduced complexity of navigation equations result in a significant reduction of computational resources needed to solve the problem. Simulations using up to 25 nodes converged in optimal solutions.

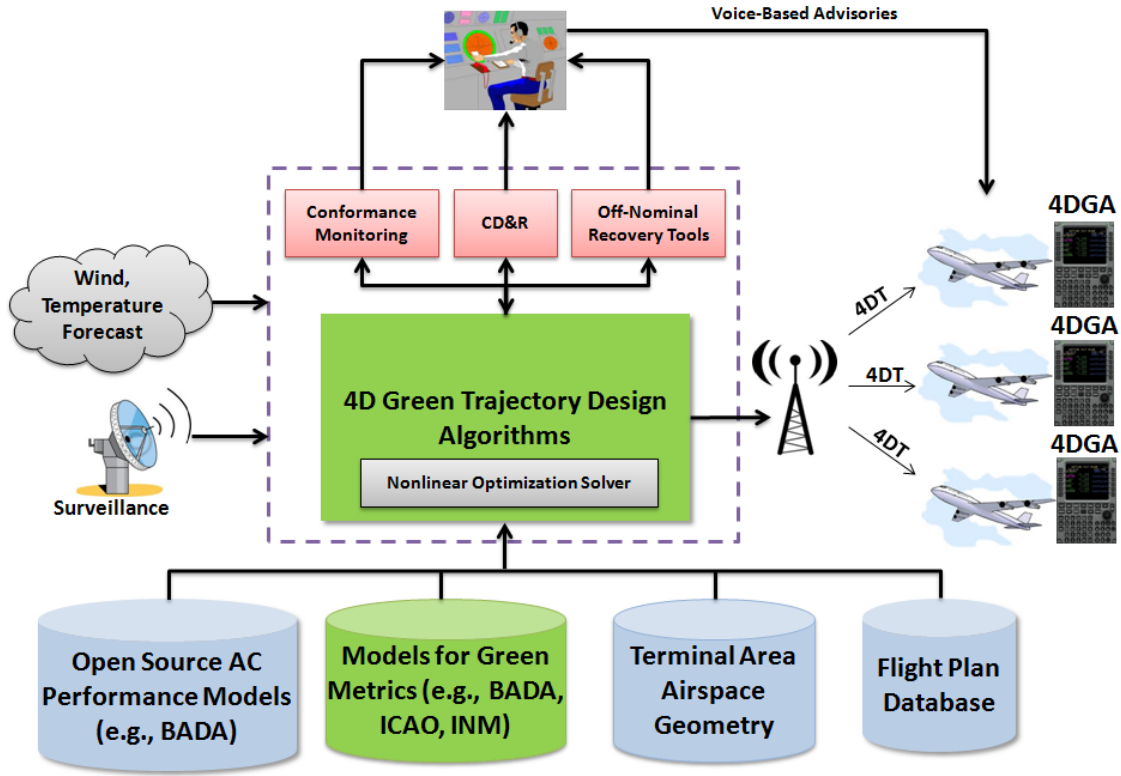
However, the results obtained show that this method introduced issues to properly follow the aircraft along the trajectory for the particular case simulated [49]. Some oscillations problems were solved by increasing the number of nodes. The guidance system has been capable to track the trajectory to fulfill the mission's requirements, however, the accuracy obtained was not the better. The results shows some difficulties to fulfill the time restrictions imposed to the performance index specification.

### **2.6.2 4D Trajectory-based concept for Terminal Area Operations**

Vaddi et al. [45], describe a unique 4D-Trajectory-based operations concept that consists in both air and ground automation systems. The ideas proposed by the authors are focused in a general systems improvement. For the particular case

of 4D trajectory generation, a method is proposed based on an optimal control problem.

The full schematic is shown in the following figure [45],



**Figure 2-4:** Schematic of 4D Trajectory-based concept [45]

In this concept, the 4DT generator is performed on ground and then sent to aircraft. Flight management System (FMS) requires to generate guidance commands to allow aircraft to track the 4D trajectory proposed by ground systems.

The system proposed by Vaddi et al. [45] uses a general performance index specification that penalizes the delay, the fuel consumption and the landing time of aircraft by the following expression:

$$\min \sum_i^N (\alpha \text{ delay} + \beta \text{ fuel}) + \gamma t_{final}$$

Components of the general expression are proposed to be calculated based on the following table:

**Table 2-6:** Performance Index Specification [45]

Objective Function	Description
Reduce Time	$t_{20}$
Reduce Fuel Consumption	$m_{fuel} = \int_0^{t_{20}} \dot{m}_{fuel}(V_{1...20}, T_{1...20}) dt$

Consequently, the equations expressed in Table 2-6 are subject to constraints defined by aircraft limits and flight profiles defined in [45]. The mathematical model proposed by the author to describe aircraft dynamics is composed by four states variables:

$$\dot{x} = V \cos(\gamma)$$

$$\dot{h} = V \sin(\gamma)$$

$$\dot{V} = \frac{(T \cos(\alpha) - D)}{m} - g \sin(\gamma)$$

$$\dot{\gamma} = -\frac{g}{V} \cos(\gamma) + \frac{L}{mV} + \frac{T \sin(\alpha)}{mV}$$

Where  $x$  is longitudinal position,  $h$  is altitude,  $V$  is airspeed,  $\gamma$  is flight path angle,  $T$  is thrust,  $L$  is lift force,  $D$  is drag force,  $m$  is aircraft mass and  $\alpha$  is the angle of attack. Obviously, this model is restricted and limited by a set of bounds and constraints that are attached to aircraft dynamics limits.

No further information is provided about the method used to discretize the dynamic equations, however, the nonlinear programming problems has been solved using the function *fmincon* of MATLAB®.

Simulations executed by the author for single and multiple aircraft scenarios show that it has been achieved realistic trajectories with the design described. For

single trajectories, it is shown a balance between time vs fuel optimization where there exists a time of arrival for which fuel consumption is minimum.

### 2.6.3 Optimal trajectories generation for next generation FMS

Diaz et al. [51] describe an optimal control approach to generate RNP, fuel-efficient 4D trajectories to be in line with requirements of next generation flight management systems.

The flight path is defined by a set of 4D waypoints composed by longitude, latitude, altitude and required time of arrival. The problem is defined by a set of RTA and RNP constraints. Required Time of Arrival is only defined at a finite number of waypoints while RNP condition is defined along the whole flight plan.

The aircraft dynamics are defined by a three-degrees of freedom, rigid-body model composed by six states variables: north distance ( $N$ ), east distance ( $E$ ), altitude ( $h$ ), true airspeed ( $V$ ), flight path angle ( $\gamma$ ) and yaw angle ( $\chi$ ).

The problem is defined by an optimal control problem that find the control vector that minimizes the following performance index:

$$J(u) = \frac{1}{2} \int_0^T (u^T R u + (y - r)^T Q (y - r)) dt + \frac{1}{2} (x(T) - x_d)^T F (x(T) - x_d)$$

Subject to the conditions:

$$\dot{x} = f(x, u)$$

$$x(t_0) = x_0$$

$$\|y(t) - r(\tau)\| < d, \quad t \in [0, T] \text{ and } \tau \in [0, T]$$

where  $R$ ,  $Q$  and  $F$  are positive defined matrix that are tuned depending of desired flight profiles or performance index conditions.

The first term of the performance index specification  $u^T R u + (y - r)^T Q (y - r)$  reduces the control effort while maintaining the aircraft position within the

trajectory for a given RNP condition. The second term  $(x(T) - x_d)^T F (x(T) - x_d)$  ensures the time of arrival for each waypoint.

The optimal control problem is solved using MATLAB® by making use of a method based on Monte Carlo simulations. The author states that this method takes around 1.41 seconds to compute the gain matrices, linearize the model and simulate the trajectory. Additionally, simulations took around 1000 iterations for a flight segment of 180 seconds.

#### 2.6.4 4D trajectory design in presence of contrails

Soler et al. [55] describes a 4D trajectory generation problem in presence of contrails. The problem is presented as an optimal control problem that minimizes the overall flying cost of fuel consumption, CO<sub>2</sub> emissions, flight time, and persistent contrails formation.

The problem is defined as an optimal control problem that minimizes the following performance index specification:

$$J(u) = C_t t_f + (C_F + C_{CO_2}) \left[ \sum_{q=0}^n \int_{t_q}^{t_{q+1}} \dot{m}_q(t) dt \right] + D$$

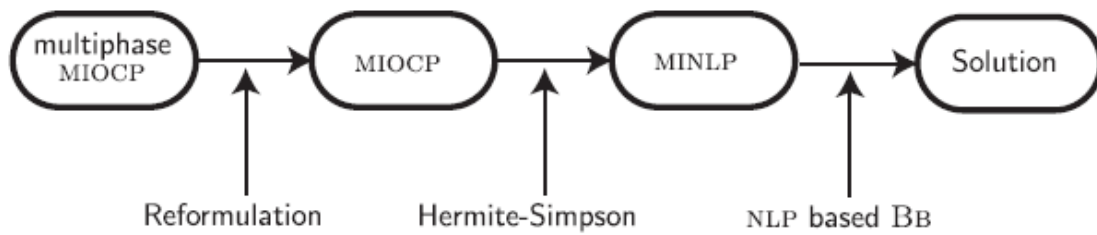
Where  $C_t$  represents flight time cost,  $C_F$  represent the fuel cost,  $C_{CO_2}$  represents the cost associated to producing CO<sub>2</sub> emissions.  $\dot{m}_q$  represents the aircraft fuel flow during a particular phase denoted as  $q$ . Finally,  $D$  represent the persistent contrail formation cost which form is defined in [55].

The model used by the author is a three-degrees of freedom that considers only aircraft forces and it is composed by seven (7) states variables: velocity ( $V$ ), heading ( $\psi$ ), path angle ( $\gamma$ ), altitude ( $h$ ), latitude ( $\theta$ ), longitude ( $\lambda$ ) and mass ( $m$ ). Additionally, wind is calculated by using polynomial regression method applied on a weather numerical model downloaded from National Oceanic and Atmospheric Administration (NOAA).

Additionally, the problem is restricted by a set of constraints and boundaries conditions denoted by  $g(x(t), u(t), t) < 0$  and  $h(x(t), u(t), t) = 0$ . Specific numeric values for each state and control variables are found in [55].

The process to solve the problem is formulating a mixed integer optimal control problem (MIOCP) by describing a set of differential-algebraic functions that represent dynamic subsystems. Subsequently, the problem is reformulated and expressed as a conventional mixed integer optimal control problem which is discretized using collocation method and finally converted into a mixed-integer nonlinear programming problem (MINLP).

The MINLP is solved using a branch and bound algorithm which is implemented in MATLAB®. The process to solve the problem by Soler et al. [55] is shown in the Figure 2-5.



**Figure 2-5:** Schematic of solution proposed by [55]

Simulations performed by the author based on a defined flight plan show that method used are valid to solve the given problem, however, the use of a very complex aircraft dynamics model introduced significant delays in finding optimal solution. The author [55] proposed using a simpler aircraft model including only five (5) states variables and neglected the wind model to improve the computational efficiency.



## 2.7 Tracking & Guidance

Several authors have focused their research in developing innovative tracking and guidance systems. The purpose of this section is to describe existing research about Tracking & Guidance systems.

### 2.7.1 4D Green Guidance for Terminal Area Operations

Vaddi et al. [44] develop a tracking and guidance algorithm based in 4D trajectories which uses green trajectories designed for terminal area operations as reference path.

The 4D Guidance Module receives a reference trajectory composed by x, y, z coordinates plus time of arrival. In addition, the module receives wind, temperature and time histories about flap position, landing gear, descent rate and aircraft airspeed.

The guidance system generates a set of control variables composed by pitch, bank angle, and thrust/throttle. The module also output the settings of the flaps and landing gear.

The 4D guidance law design uses an aircraft model composed by the equation of motion, the engine model, the pitch, the controls and the external inputs. All the equations used in this research are restricted to the vertical plane.

The equation that describe the dynamics of the aircraft are composed by the equation of motion in the vertical plane, the engine equations, an autopilot model, control variables and external inputs. The full model description can be found at [44].

A standard atmospheric model is used to compute the density and temperature as a function of the aircraft altitude.

The altitude relative to the standard atmosphere conditions (density altitude) is represented as a function of the density and the temperature as follows [44]:

$$h_d = \frac{T_o}{a_t} \left[ 1 - \left( \frac{\rho}{\rho_o} \right)^{\left( \frac{a_t R}{g + a_t R} \right)} \right]$$

where  $T_o$  is the standard temperature at sea level,  $a_t$  is the temperature gradient (which is constant for values below 36,000 ft.)  $\rho_o$  is the standard density at sea level and  $g$  is the gravity acceleration.

The trajectory design also consider the prediction of the wind effect in the aircraft dynamics.

The recorded data is used to simulate the nominal wind as a function of North, East, and altitude cartesian coordinates plus time.

The proposed Tracking and Guidance system is shown in the Figure 2-6. The system is composed by the following blocks [44]:

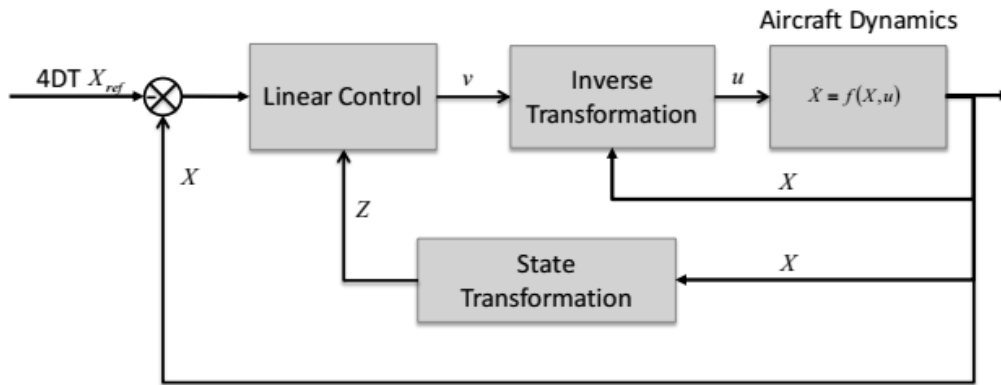


Figure 2-6: Control Loop proposed by Vaddi et al. [44].

The primary purpose of the state transformation is to express the system as a set of linear states with time-invariant parameters. This linear system is controlled by defining a set of pseudo-control variables ( $v_1$  and  $v_2$ ).

The linear control block is a proportional controller to compute the control variables mentioned ( $v_1$  and  $v_2$ ) plus an integrator state introduced in order to reduce the steady-state errors. The gains were adjusted empirically after some experiments.

This block provides the inverse transformation of the pseudo controls inputs. This will be achieved only if the matrix  $g_{ij}$  is invertible.

The control inputs define the response of the dynamics of the aircraft. This block computes the values of the aircraft states that are used as an output (and feedback signal).

### 2.7.2 4D Tracking System for Waypoint-Based Navigation

Bousson and Machado [49] initially described a process to generate a 4D trajectory (section 2.6.1). Subsequently, they have developed a tracking system to evaluate the trajectory generated.

The aircraft dynamics model is described by eight (8) states: velocity ( $V$ ), flight path angle ( $\gamma$ ), heading ( $\psi$ ), bank angle ( $\phi$ ), pitch angle ( $\theta$ ), roll ( $p$ ), pitch ( $q$ ) and yaw ( $r$ ) rates.

Their guidance system is based on the theory of one-step predictive control where the model is represented as a combination of two subsystems:

$$\dot{x}_1 = f_1(x)$$

$$\dot{x}_2 = g(x, u)$$

The two equations are approximated<sup>1</sup> using Taylor series expansion in order to obtain a discretised system so that the expressions  $x_{1k+1}$  and  $x_{2k+1}$  are defined:

$$x_{1k+1} = x_{1k} + \Delta\tau f_1(x_k) + \left(\frac{(\Delta\tau)^2}{2}\right)[F_1 f_1(x_k) + F_2 f_2(x_k) + B(x_k)\overline{u_k}]$$

$$x_{2k+1} = x_{2k} + \Delta\tau(f_2(x_k) + B(x_k)\overline{u_k})$$

where

$$F_1 = \left(\frac{\partial f_1(x)}{\partial x_1}\right)_{x=x_k}$$

$$F_2 = \left(\frac{\partial f_2(x)}{\partial x_2}\right)_{x=x_k}$$

---

<sup>1</sup> More detailed description of the approximation method is available in [49]

The problem is solved as an optimization problem, where the objective function is designed in order to minimize the difference between a reference vector  $X_{ref}$  and the current position of the aircraft  $X_{k+1}$ .

$$J(u) = \frac{1}{2} e_{1k+1}^T Q e_{1k+1} + \frac{1}{2} e_{2k+1}^T Q e_{2k+1}$$

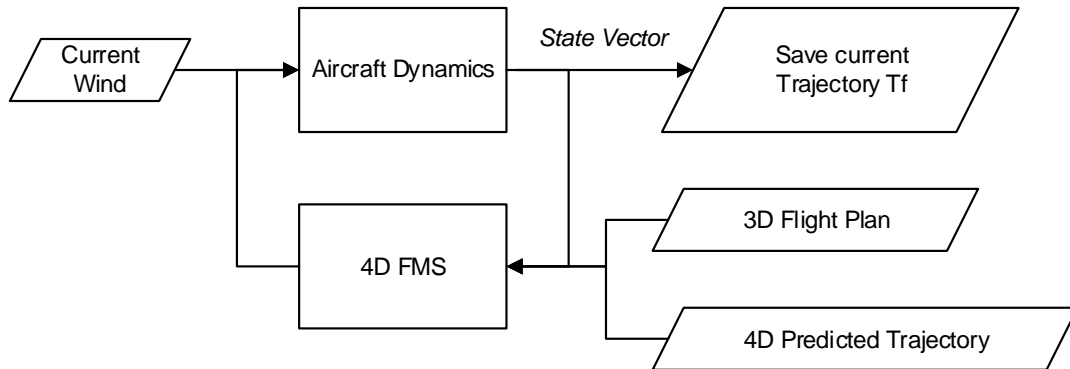
where  $e_1$  and  $e_2$  are the difference between the current track and the aircraft position for each instance of  $k$ .

### 2.7.3 4D Descent Trajectory Guidance

Canino et al. [21] develop a 4D tracking & guidance system based in a conventional 6-degree of freedom point mass model for descent phase with acceptable accuracy at low computational cost.

The errors computed for the tracking phase are the maximum along track deviation, the maximum altitude track deviation respect to foreseen vertical descent profile and the maximum lateral deviation.

Their guidance system is shown in Figure 2-7:



**Figure 2-7:** Guidance and Control System Proposed by Canino et al. [21]

The 4D FMS guidance system computes command values for thrust, bank angle and path angle using the predicted trajectory and the current state vector provided by the aircraft dynamic model.

The lateral and vertical guidance in the 4D FMS model described is based on traditional LNAV and VNAV methods of 3D FMS. The overview of the system proposed by Canino et al. is shown in Figure 2-8. First, it is visible that an estimated 4D trajectory is joined to the flight plan states to compute longitudinal guidance commands and maintain the altitude inside the predicted aircraft 4D trajectory. Second, lateral guidance commands are only computed in case aircraft is in descent phase.

The along track guidance algorithm used in this system corrects and computes the along track deviation ( $dl$ ) at each instant  $i$  and is defined as follows:

$$dl = (x_{fi} - x_{0i}, y_{fi} - y_{0i}) u_i$$

Where  $x_{fi}$  and  $y_{fi}$  is the final position of aircraft at instant  $i$ . Tracking  $dl$  is performed using speed changes by computing a new true airspeed in order to maintain the ground vector using the following expression:

$$TAS_f = (GS^2 + W_f^2 - 2 GS_0 W_f \cos B_f)^{\frac{1}{2}}$$

Where  $W_f$  is the current wind direction,  $GS$  is the ground speed and  $B_f = |\omega_{h\_f} + \varphi|$  and  $\omega_{h\_f}$  is the current wind direction.

The tracking  $dl$  can also be achieved from the lateral deviation. This procedure consists in creating a lateral path that allows the aircraft to increase the descent profile. The lateral paths are computed using new waypoints inserted on both sides of the route.

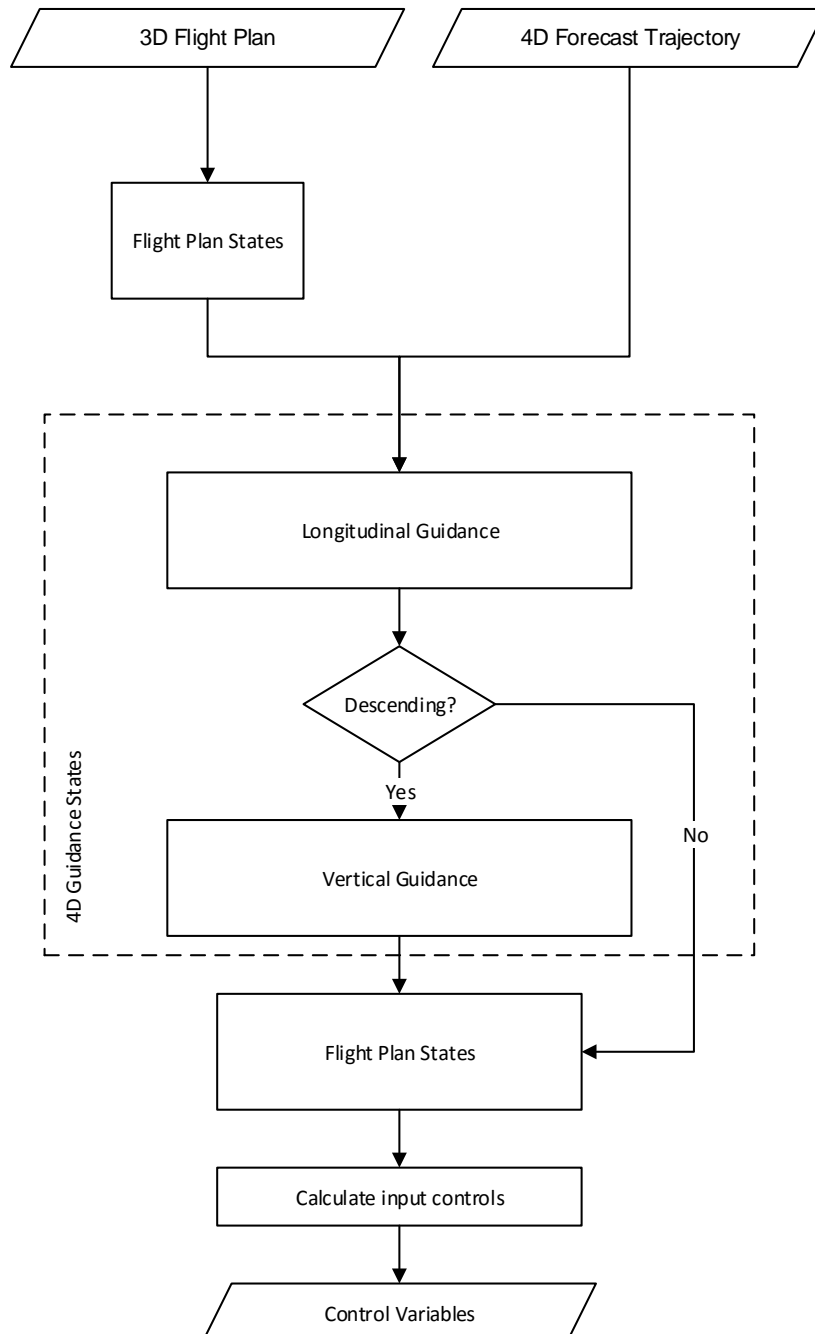
The distance error could be calculated using the following expression:

$$dl = \frac{dl + \Delta_{turn}}{2 (1 - \cos(\alpha))}$$

where  $\Delta_{turn}$  is the dynamic turn rate and  $\alpha$  is the angle between aircraft and the interception point.

If the reference path is below the current aircraft position and thrust is at idle position, it is necessary to compute a new drag value to increase the rate of descent.

$$D_f = k_D * D_0$$



**Figure 2-8:** Overview of system proposed by Canino et al. [21]

where the factor  $k_D$  is defined by:

$$k_D = 1 + \frac{mg}{D_0 TAS ESF} \Delta \dot{z}$$

Tracking  $dh$  can be also achieved by a lateral deviation by computing a new waypoint. This procedure is similar to the one used for lateral deviation while tracking the longitudinal error in the along-track guidance.

## 2.8 Optimal Control Methods

Optimal control problems can be solved using different strategies and methods. Direct methods has been widely applied in trajectory optimization problems. Various discretization methods can be applied in order to convert the problem into a Nonlinear Programming (NLP) problem. Numerous solvers are available to solve the NLP problem.

Bouson and Machado [49] use pseudo spectral parametrization of time dependent variables to discretize the problem. The time-dependent variables are discretized by making use of Chebyshev and Lagrange polynomials. The process to approximate the variables is based on Legendre method built on Chebyshev nodes defined, as usual, in the interval  $[-1, 1]$ . Then Lagrange interpolating polynomials are defined based on Chebyshev nodes.

Once problem is discretized, the problem is converted into a Nonlinear Programming (NLP) problem. The MATLAB® function *fmincon* is used to solve the resultant NLP problem and obtain an optimal solution.

Some significant oscillations in one of the control variables have been detected in the results. This problem has been solved by the author while increasing the number of nodes used in the pseudo spectral parametrization.

Soler et al. [55] implement a discretization method of states and control variables by applying a collocation method based on Hermite-Simpson Gauss-Lobatto quadrature rules. They convert the problem into a mixed integer nonlinear programming problem (MINLP).

Furthermore, this problem is solved by using a branch and bound algorithm described in [55]. In order to use the Bonmin open-source solver, the problem is modelled using AMPL modelling language.

Geiger et al. [17] describe a technique to generate optimal trajectory for an UAV using Direct Collocation with Nonlinear Programming (DCNLP). The problem is discretized using direct collocation method based on third-order Hermite polynomials. Subsequently, the problem is converted into a NLP problem and solved by MATLAB® function *fmincon*.

Tian and Zong [40] suggest an adaptive Gauss pseudo spectral method to optimize an ascent phase aircraft trajectory. The problem has been solved using Sparse Nonlinear Optimizer (SNOPT®).

Grüning et al. [46] use pseudo spectral optimizer (PSOPT) to solve an optimal control problem in the design of a feedforward control for a four-rotor UAV.

Most researches about optimal trajectories generation use MATLAB® *fmincon* function to solve nonlinear programming problems. The flexibility that offers MATLAB® is considerable with respect to other software. However, there are many other discretization methods that several authors have applied to solve optimal control problems for applications different than aircraft trajectory optimization. Consequently, many researches in aerospace applications make use of other tools different than MATLAB® such as IPOPT, SNOPT®, PSOPT, and DYNOPT.

Additionally, other authors [51] [42] use stochastic methods such as Monte Carlo simulations implemented in MATLAB® or Genetics Algorithms (GA) implemented in stand-alone Java applications.

## **2.9 Conclusion**

Several projects have been developed in order to generate 4D optimal trajectories. Most of previous authors used an optimal control theory approach to find the best solution. The approach of Bouson et al. [49] has the advantage of using a waypoint-based optimization that could be applied to any flight plan.



However, the use of aircraft dynamics instead of navigation equations will ensure the aircraft is capable to fly the optimized trajectory. Similar characteristics of simplicity are shown in Vaddi et al. [45] research. The optimization process and mathematical model was designed for vertical profile. The use of a 3D model is required so that aircraft trajectory is optimized in vertical and horizontal profile. In contrast, using complex models rises the number of states variables and consequently the execution time when solving the optimal control problem is increased.

Different approaches can be followed to design the performance index specification. In the particular case of Soler et al. [55] optimization was carried out based on total flight cost by adding fuel cost, time cost and CO<sub>x</sub> emissions cost in the same performance index specification. However, the use of this approach requires to properly balance the total equation, for example, by making use of weights factors.

Most authors used direct methods to solve the optimal control problem. The time-dependent variables are discretized and then the resultant non-linear programming (NLP) problem is solved using a sparse matrix solver. The most common NLP solver used is *fmincon* included in MATLAB software. Despite the simplicity of this method, it is not suitable for stand-alone applications given that it requires third-party compilers. In addition, there exist other methods that are free and open source. It is required to use an optimal control solver capable to be used in stand-alone applications.

# ***Chapter 3***

## **4D TRAJECTORY & GUIDANCE OPTIMIZATION TECHNIQUES**

### **3.1 Introduction**

This chapter describes optimization techniques which are used to develop the 4D Trajectory Generator & Guidance system. The optimal control problem (OCP) is described in section 3.2. Section 3.3 describes the methods to solve OCP. Section 3.4 describes and compares the different software tools for solving OCP and section 3.5 describes the concept of proportional navigation.

### **3.2 Optimal Control Problem**

Optimal control is defined as the process of determining control and states trajectories for a dynamic system over a period of time to minimise a performance index [36].

In order to solve an optimal control problem, it is necessary to identify:

1. A mathematical model of the system to be optimized
2. A cost function or specification of performance index
3. Boundaries and constraints specifications for system variables

Our objective is to seek an optimal control law vector  $u$  which minimises the expected total cost of a system for starting at a given  $[x(t_0); t_0]$  and ending at  $[x(t_f); t_f]$ .

Therefore, the optimal control problem can be expressed as minimize the cost functional (also known as performance index  $J(x, u)$ ):

$$J(x, u) = \varphi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) dt \quad (3-1)$$

in order to find the control vector  $u[t_0, t_f]$  subject to the dynamic constraints:

$$\dot{x}(t) = f(x(t), u(t), t), x(t_0) = x_0 \quad (3-2)$$

This problem is generally known as *Bolza* problem [36]. If the performance index takes the form:

$$J(x, u) = \int_{t_0}^{t_f} L(x(t), u(t), t) dt$$

it is known as the *Lagrange* problem [36]. Similarly, if performance index takes the form:

$$J(x, u) = \varphi(t_0, x(t_0), t_f, x(t_f))$$

the problem becomes the *Mayer* problem [36].

The problem formulation is completed by defining a set of boundaries and constraints as follows:

$$g(x(t), u(t), t) \leq 0$$

$$h(x(t), u(t), t) = 0$$

Complex optimal control problems cannot be solved using analytical approaches. For this reason, extensive use of computers supposes an important progress for optimal control.

### 3.3 Direct Methods

The traditional analytical approaches cannot solve real-world complex optimal control problems, such as 4D trajectory optimization problems. Many numerical methods have been investigated to solve optimal control problems over the last few decades, including direct solution methods and indirect solution methods.

indirect methods rely in the *Pontryaing's Maximum Principle* [1]. These methods attempts to find a minimum point indirectly, by solving the necessary conditions of optimality in an optimal control problem. The problem is converted into a boundary value problem.

Most common numerical methods used to deal with boundary value problems are Gradient-based, Shooting-based (Single and Multiple) and Indirect Collocation methods.

Despite of their high accuracy, indirect methods solutions are very sensitive to small changes, which is a disadvantage when dealing with high constrained problems.

Additionally Indirect methods convergence depends on the good estimation of initial values. Another important aspect is that they are hard to apply when there exist path constraints or singular arcs.

In contrast, a direct method attempts to find a minimum to the cost function by constructing a sequence of points converging to that minimum. The main idea of direct methods relies on discretizing the problem (usually control and state variables) in order to convert the system into a Nonlinear Programming (NLP) problem. [37].

The general NLP problem tries to find the  $n$ -vector  $x^T = (x_1, \dots, x_n)$  that minimizes the function [37]

$$F(x) \tag{3-3}$$

subject to the constraints

$$c_L \leq c(x) \leq c_U$$

and the simple bounds

$$x_L \leq x \leq x_U$$

Direct methods are appropriate to deal with problems involving path constraints with relatively low computational cost [16].

### 3.3.1 Direct Collocation

Direct collocation methods aims at finding the best solution of a set of candidate solutions denoted by polynomials, so that optimal solution is satisfied at certain points called *collocation* knots. Depending on how collocation points are chosen, the collocation method could be: orthogonal or standard [29].

Orthogonal collocation methods rely in the use of quadrature nodes to approximate continuous functions. Quadrature rules are often determined by Legendre or Chebyshev polynomials in famous orthogonal collocation methods [29] [11] [70]. It is known that quadrature produces accurate results if the function  $f(x)$  is approximated in the interval  $[-1, 1]$ .

Pseudo spectral collocation methods are suitable for approximating smooth functions, integrations, and differentiations [36], for this reason its use in optimal control problems has been extended.

The use of Legendre polynomials are popular in collocation methods. Using these polynomials for solving differential equations in optimal control problems was proposed by Elnagar et al. in 1995 [11].

A Legendre polynomial is orthogonal over the interval  $[-1, 1]$  and it is generated from:

$$L_N(\tau) = \frac{1}{2^N N!} \frac{d^N}{d\tau^N} (\tau^2 - 1)^N \quad (3-4)$$

where

$$\tau \leftarrow \frac{2}{t_f - t_0} t - \frac{t_f + t_0}{t_f - t_0} \quad (3-5)$$

Some examples of Legendre polynomials are shown in [70]:

$$L_0(\tau) = 1$$

$$L_1(\tau) = \tau$$

$$L_2(\tau) = \frac{1}{2}(3\tau^2 - 1)$$

$$L_3(\tau) = \frac{1}{2}(5\tau^3 - 3\tau)$$

If  $L(\tau)$  is a general smooth function, then its integral over  $\tau \in [-1, 1]$  can be approximated as follows:

$$\int_{-1}^1 L(\tau) d\tau \approx \sum_{k=0}^N L(\tau_k) \omega_k \quad (3-6)$$

where  $\omega_k$  are weights given by:

$$\omega_k = \frac{2}{N(N+1)} \frac{1}{[L_N(\tau_k)]^2} \quad (3-7)$$

The pseudo spectral Legendre Method is explained in [11]: “Given the function  $F(t)$  defined over  $[-1, 1]$  we construct its  $N^{\text{th}}$  degree interpolating polynomial as follows: Define the Lagrange polynomials

$$\phi_l(t) = \frac{2}{N(N+1)} \frac{1}{L_N(t_l)} \cdot \frac{(t^1 - 1)L_N(t)}{t - t_l}, (l = 0, 1, \dots, N) \quad (3-8)$$

The  $N^{\text{th}}$  degree interpolation polynomial,  $F^N(t)$ , to  $F(t)$  is given by:”

$$F^N(t) = \sum_{k=0}^N F(t_k) \phi_k(t) \quad (3-9)$$

Elnagar et al. [11] explain that  $F^N(t)$  can be obtained differentiating the equation (3-9) and the result is a matrix multiplication given by:

$$\dot{F}^N(t_m) = \sum_{l=0}^N D_{ml} F(t_l) \quad (3-10)$$

where  $D$  is a  $(N+1) \times (N+1)$  matrix (defined in [11]).

### 3.4 Tools for Solving Optimal Control Problems

Several tools based on direct methods have been developed to solve optimal control problems. Many of them focus on solving NLP problem while allowing the user to choose the most appropriate method to discretize the problem.

SNOPT<sup>®</sup> is a popular sparse nonlinear programming solver written in FORTRAN by Gill et al. [71]. The solver has been written mainly in FORTRAN but it has been released in C, C++ and MATLAB<sup>®</sup> versions.

SNOPT<sup>®</sup> is based on sparse SQP algorithm based on quasi-Newton approximations to the Hessian of Lagrangian. SNOPT<sup>®</sup> is distributed with proprietary license [71]. It is commercially available with different licenses for commercial, student and single use.

It is used by several applications and software packages due to its considerable stability. Some of these software packages are AeroSpace Trajectory Optimization & Software (ASTOS) [75], General Mission Analysis Tool (GMAT) [76], and Optimal Trajectories by Implicit Simulation (OTIS) [77].

Another important aspect of this nonlinear programming solver are its capabilities to support modelling languages such as “A Mathematical Programming Language” (AMPL) which is an algebraic modelling language for describing complex mathematical problems.

Another software tool for solving NLP is Interior Point Optimizer (IPOPT) [67]. It is a large-scale nonlinear programming solver written in C++ and distributed as Eclipse Public License (EPL). The code has been written by Wächter and Laird.

IPOPT is very famous due to its extensive environments where it can be used on. Linux, Windows<sup>®</sup> and Mac OS X and with versions in Java, MATLAB<sup>®</sup> and FORTRAN are examples of IPOPT flexibility.

Some other tools such as DYNOPT [62] also provide capabilities for discretizing the time dependent variables and solving the problem using one of the popular nonlinear programming solvers.

The problem is solved using total discretization with orthogonal collocation on finite elements. The software uses Lagrange interpolation polynomials to approximate states and control vectors. The algorithm used to develop this tool is based on the work of Biegler et al. [7].

An important feature of DYNOPT is the capability to integrate with the popular MATLAB® Optimal Control Toolbox and making use of the function *fmincon*. Therefore, DYNOPT requires MATLAB to work since it has been implemented using this proprietary software.

Pseudo Spectral OPTimal Control Solver (PSOPT) is a C++ framework delivered as a set of libraries that have been developed by Becerra [70].

PSOPT uses direct collocation methods including pseudo spectral and local discretization. Pseudo spectral discretization methods use Chebyshev or Legendre functions to approximate the time-dependent variables. Local discretization approximates the time dependent functions using splines. The problem is solved using total discretization with orthogonal collocation on finite elements.

PSOPT is free and distributed under GNU license; it can be compiled on GNU Linux or Microsoft® Windows® 7. Currently, there exists some installers that speed-up the installation process.

There are several advantages of using PSOPT. Its flexibility for using the most important nonlinear programming solvers (IPOPT or SNOPT) is a powerful capability thus it allows the user to compare results of the same optimal control problem by just making few changes in the source code (switching NLP solver). Another important aspect is the capability to utilize different discretization methods.

The National Aerospace laboratory of Netherlands (NLR) has included PSOPT as a key tool for optimizing trajectories in order to reduce fuel and noise [70]. This has been carried out as part of NLR activities involved in CleanSky European Project.



Additionally, it is used to develop optimal trajectories in the project ASTER [43], a Brazilian mission to asteroid “2001 SN263” planned to be launched in 2016.

PSOPT structure is described in the provided user manual [70].

The following table summarizes and compares different optimal control tools with respect to PSOPT:

**Table 3-1:** Comparison between Optimal Control Tools

<b>Tool</b>	<b>Method</b>	<b>Language</b>	<b>License</b>	<b>Details</b>
<b>SNOPT</b>	SQL	FORTTRAN, C++, MATLAB®	Proprietary	Control Problem requires to be converted to NLP
<b>IPOPT</b>	NLP	C, C++, Java, FORTTRAN, MATLAB®	Free	Control Problem requires to be converted to NLP.
<b>DYNOPT</b>	Collocation	MATLAB®	Free	Uses Control Toolbox
<b>PSOPT</b>	Collocation (Legendre)	C++	Free	Uses IPOPT or SNOPT
<b>Control Toolbox</b>		MATLAB®	Proprietary	

As shown in Table 3-1, some tools are proprietary license. This makes difficult its use into a stand-alone application since there is not access to their source code.

Despite their high capabilities in optimal control, other applications such as DYNOPT, requires third-party compilers provided by MATLAB® to create executable files, which could be an important limitation when trying to run in other computer.

PSOPT seem to be a suitable option for using into a stand-alone application since they are free open source and provides powerful performance while being implemented in C++. Another important advantage is that PSOPT can be setup to use two different non-linear programming solvers: IPOPT or SNOPT. This is considered an useful feature to compare and evaluate the results.

Other applications were neglected for this comparison because they were considered stand-alone software with limited integration capabilities.

### **3.5 Guidance using Proportional Navigation (PN)**

Proportional Navigation (PN) is a concept that was introduced in 40's in the United States as a control law used for missile guidance [13]. In second World's War, some steps towards were implemented by German scientists. In the 50's, the use of PN for missile guidance was considerably extended. Currently there exists numerous variations of the proportional navigation.

Despite new advances in Control Theory achieved in recent years, proportional navigation is still used in most modern missile guidance systems. Example of this is AIM-9 Sidewinder [57]. The main reasons of this, are that proportional navigation is cheap to implement and has demonstrated effective guidance in missiles.

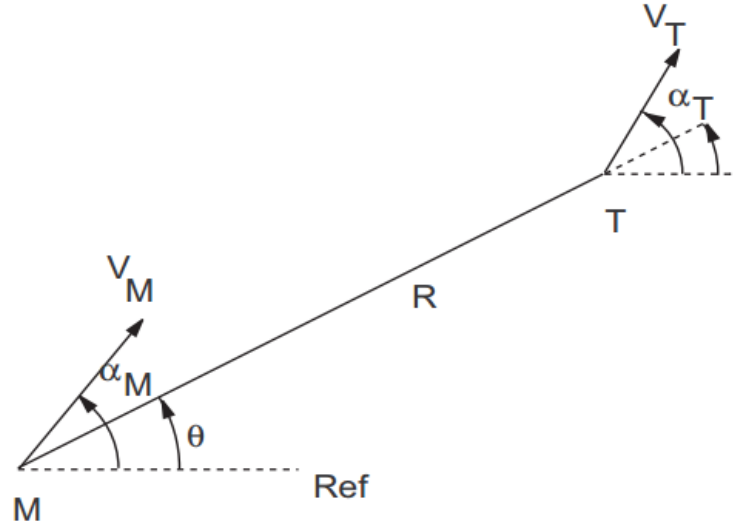
Proportional navigation has some limitations related to evasive manoeuvres when the pursuer is close to target. For example, proportional navigation becomes considerable less effective if evader performs fast accelerations or zig-zag movements when missile is approaching.

The basic idea behind proportional navigation is to generate missile acceleration commands in proportion to its line-of-sight (LOS).

Figure 3-1 shows the two-dimensional engagement geometry of a missile which pursues a non-maneuvering target.

The equations of motion of this system based on Point-Mass-Model (PMM) can be found in [54].

The proportional navigation (PN) guidance law focuses on finding a guidance command that ensures that missile rate of rotation is proportional to target rate of rotation (based on missile line-of-sight (LOS)).



**Figure 3-1:** Missile-target engagement geometry [54]

This is equivalent to:

$$\dot{\alpha}_M = N\dot{\theta}$$

where  $N$  is the navigation constant.  $\alpha$  and  $\theta$  are shown in Figure 3-1 above.

In addition, there exists enhanced PN-based guidance laws such as Pure Proportional Navigation (PPN), True Proportional Navigation (TPN) and Ideal Proportional Navigation (IPN) [54].

Most notable research about Proportional Navigation were carried out by Guelman [3] in 1971 and extended by Ghawgawe and Ghose [12] in 1996.

It is well-known that Pure Proportional Navigation (PPN) has been mostly extended on missile guidance applications. However, True Proportional Navigation (TPN) has been extended into several other fields such robotics, space travel, spacecraft landing and aeronautical applications due to its wide availability of literature.

There exists several applications of Proportional Navigation (PN) to aeronautics and guidance systems. Smith [25] used PN with adaptive terminal guidance for aircraft rendezvous. Han and Bang [56] applied Proportional Navigation (PN) in the design of an Avoidance Collision strategy for Unmanned Air Vehicles. Yamasaki and Balakrishnan [33] introduced a high performance pure pursuit guidance method for UAV rendezvous and chase with a cooperative aircraft.

### 3.6 Other Concepts

This section describes other concepts that are considered useful topics to understand this thesis.

#### 3.6.1 Andrew's Monotone Chain Convex Hull Algorithm

Convex Hull algorithms are used to find convex polygons based on a list of two dimensional points. This section gives an overview of Andrew's Monotone Chain Convex Hull algorithm [6].

The algorithm starts on a basis that all points are initially sorted by their  $x$  coordinate.

Then, there are selected tow points,  $P_a(x_a, y_a)$  and  $P_b(x_b, y_b)$  so that:

$$x_a = \min(X)$$

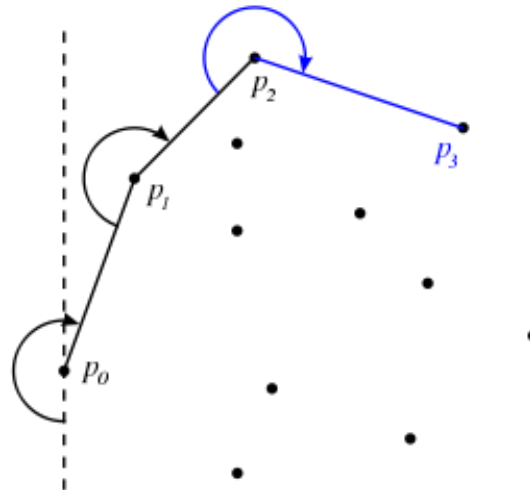
$$x_b = \max(X)$$

where  $X$  is the domain of size  $n$  that contains all  $x$  coordinate values  $(x_1, x_2, x_3 \dots x_n)$ .

Then the set of points can be divided into two different lists depending of the point position with respect to the vector  $\overline{P_a P_b}$ . It is obtained an *upper* hull list and a *lower* hull list.

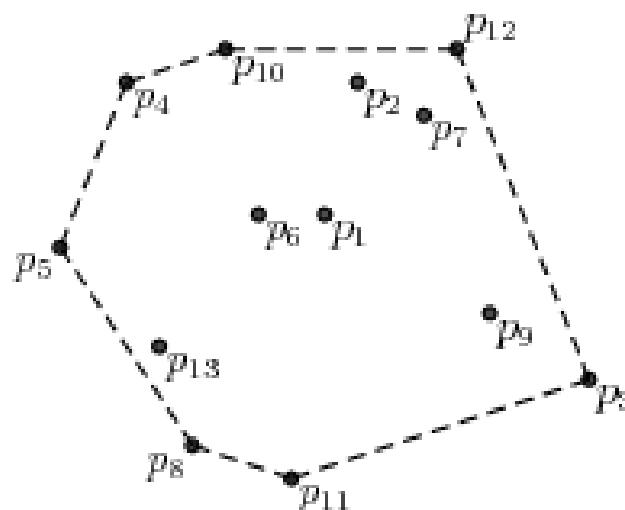
Each element is checked so if a vector between one point and next point is clockwise oriented with respect to previous vector, it is a convex angle. Otherwise the point is removed from the list.

The idea is expressed in Figure 3-2. Vector composed by points  $P_1$  and  $P_2$  is *clockwise oriented* with respect to vector composed by point  $P_0$  and  $P_1$ . Similarly vector  $\overline{P_2P_3}$  is clockwise oriented with respect to  $\overline{P_1P_2}$ . A point is removed from the list when vector orientation *turns to left* (counter clockwise) with respect to previous one.



**Figure 3-2:** Monotone Chain | Convex Angles

Once all points are checked, upper and lower hull lists are then joined. Only remaining points in both lists are the ones that creates a convex polygon as shown in Figure 3-3.



**Figure 3-3:** Convex polygon example

# Chapter 4

## 4DT GENERATOR DEVELOPMENT

### 4.1 Introduction

This chapter describes the design and development of 4D trajectories generator. It covers aircraft dynamics model for trajectory generator in section 4.2, performance index specification for reducing fuel, time and noise is described in section 4.3, and boundaries & constraints are described in section 4.4.

Furthermore 4D Trajectory Research software is introduced in section 4.5, then it is explained the software structure in section 4.6. 4DT RS Core is described in section 4.7 and graphic user interface in section 4.8, detailed information about computing flight data in section 4.10. Finally some additional tools and trajectory representation functions are explained in sections 4.11 and 4.12.

### 4.2 Aircraft Dynamics Model

Commercial aircraft trajectories usually have limited and small rotation motions. Therefore, the dynamic model used for the 4D trajectory generation is based on a 3DoF point-mass aircraft model. The model state  $\mathbf{x}$  vector consists of seven states  $\mathbf{x} = [x, y, z, V, \psi, \gamma, m]^T$ : north distance, east distance, altitude, heading, path angle, bank angle and mass. The model control vector  $\mathbf{u}$  consists of three variables  $\mathbf{u} = [\phi, \alpha, T]^T$ : bank angle, angle of attack and thrust [20].

$$\dot{x} = V \cos(\gamma) \cos(\psi) \quad (4-1)$$

$$\dot{y} = V \cos(\gamma) \sin(\psi) \quad (4-2)$$

$$\dot{z} = V \sin(\gamma) \quad (4-3)$$

$$\dot{V} = \left(\frac{g}{m}\right) [T \cos(\alpha) - D - m \sin(\gamma)] \quad (4-4)$$

$$\dot{\psi} = \left(\frac{g}{mV \cos(\gamma)}\right) (T \sin(\epsilon) + L) \sin(\phi) \quad (4-5)$$

$$\dot{\gamma} = \left(\frac{g}{mV}\right) (T \sin(\alpha) + L) \cos(\phi) - m \cos(\gamma) \quad (4-6)$$

$$\dot{m} = -\eta T C_{fcr} \quad (4-7)$$

where

$x, y$  are north and east position

$z$  is the altitude

$V$  is velocity

$\psi, \gamma, \phi$  are heading, path angle and bank angle respectively

$m$  is aircraft mass

$g$  is gravity constant

$\alpha$  is angle of attack

$C_{fcr}$  is the cruise fuel flow coefficient

$D$  is the drag and  $L$  is the lift defined as follows:

$$D = \frac{1}{2} C_D \rho S V^2 \quad (4-8)$$

$$L = \frac{1}{2} C_L \rho S V^2 \quad (4-9)$$

where  $C_D$ ,  $C_L$  and  $S$  are variables unique for each airplane and they are the drag and lift coefficients and the wing platform area respectively.  $\rho$  is the density of the atmosphere and  $\eta$  is the specific fuel consumption that is defined by BADA [78] as follows:

$$\eta = C_{f1} \left( 1 + \frac{V}{C_{f2}} \right) \quad (4-10)$$

where  $C_{f1}$  and  $C_{f2}$  are fuel coefficients unique for each aircraft.

Wind is considered an important component because of its effect in fuel consumption and estimation of time of arrival. However, experience in practice and theory shows that the use of a simple aircraft model is necessary to solve the optimal control problem when there exists limited computational resources [16] [49] [55]. This computational efficiency was necessary to perform fast tests to early versions of the synthesizer. Therefore, the wind effect has been neglected for the first version of the 4DT generator.

### 4.3 Performance index specification

Cost functions are expressed as performance index specification in order to reduce fuel, time or noise. These functions have been implemented internally. A-priori conditions could be defined by a flight profile before generating 4DT trajectories.

#### 4.3.1 Reducing Fuel and Time

The Fuel performance index is defined as follows:

$$J_{fuel}(t) = \int_{t_0}^{t_f} \left( \frac{(m(t) - m(t_0))^2}{m_{max}} + \frac{(z(t) - z_p)^2}{z_{max}} \right) dt \quad (4-11)$$

where.

$z_{max}$  is the maximum altitude

$z_p$  is the altitude at target point



$m_{max}$  is the maximum mass

$m(t_0)$  is the initial mass

The performance index  $J_{fuel}(t)$  attempts to maintain the initial mass  $m(t_0)$  while following the reference altitude  $z_p$ . In order to normalize the scale of performance index, each element of equation (4-11) is divided by maximum mass  $m_{max}$  and maximum altitude  $z_{max}$  constants.

Time performance index is defined as follows:

$$J_{time}(t) = \int_{t_0}^{t_f} \left( \frac{(V(t) - V_{max})^2}{V_{max}} + \frac{(z(t) - z_p)^2}{z_{max}} \right) dt \quad (4-12)$$

where,

$V_{max}$  is the maximum aircraft velocity

The performance index  $J_{time}(t)$  attempts to maintain aircraft maximum velocity  $V_{max}$  while follows the reference altitude.

Following the same approach proposed by Sabatini et al. [53], once defined a flight profile, scaling factors could be assigned in order to balance performance index function to selected requirements. Some test were performed using different scaling factors.

### 4.3.2 Reducing Noise

*Sound Exposure Level (SEL)* is defined as a logarithmic measure of the sound pressure perceived at a reference location. This reference location is addressed as *observer*.

Predicting noise produced by an aircraft is an extremely complex task. Currently, there no evidence of existing continuous models capable to predict aircraft noise accurately. Existent models utilised by previous researchers in optimal control problems depend of engine parameters that are difficult to obtain in public domains and their accuracy has been demonstrated only on specific cases.

Additionally, noise is generated by different components. In the case of departure phase, main source of noise is generated by aircraft engine(s), which usually are set to generate maximum thrust during take-off. However, in cruise and mainly in approach phases, aircraft surfaces such as slats, spoilers, flaps and landing gear generate significant amount of noise which could comparable to the one produced by engines [8].

Empirical data has been used in order to represent aircraft engine noise. Since surfaces or any other component has not been considered to represent aircraft noise, optimization process regarding to *noise reduction* has been limited only to departure phases.

Noise specification empirical data contains different parameters such as engine model, type of measure, operation, thrust setting and sound level measured at different distances defined by distance values from 200 to 25,000 feet as follows:

NPD_ID	MET	OP	THR	Distance										C
				200	400	630	1000	2000	4000	6300	10000	16000	25000	
2CF650	E	D	25000	109.8	105.1	101.5	97.3	90.3	82.0	76.0	70.0	62.7	53.9	J
2CF650	E	D	40000	113.0	108.6	105.2	101.5	95.6	88.2	83.1	77.5	70.8	63.3	J
Engine				Thrust			Noise Levels							

**Figure 4-1:** Noise specification empirical data example

A typical Sound Exposure Level (SEL) with respect to distance for a common jet engine is shown in Figure 4-2 (blue). In contrast, green/red colour lines represent the result of approximating SEL distribution by using a piecewise-defined function:

$$SEL = \begin{cases} A_1 \log(x) + B_1 & \rightarrow 0 \leq x \leq 4000 \\ A_2 x + B_2 & \rightarrow 4000 < x \leq 25000 \end{cases} \quad (4-13)$$

For the first part of this piecewise function, if only two (2) points are given, then a logarithm expression  $A_1 \log(x) + B_1$  could be obtained by solving the equation system and obtaining  $A_1$  and  $B_1$  coefficients:

$$\begin{cases} y_1 = A_1 \log(x_1) + B_1 \\ y_2 = A_1 \log(x_2) + B_1 \end{cases} \quad (4-14)$$

However, since it is provided a set of  $m$  points, it is taken into account all available points by calculating different values of coefficients  $A_1$  and  $B_1$  for each possible pair of points. Then these values are sum and divided by the number available of points to obtain an average value of  $A_1$  and  $B_1$ . This process is summarized in the following equations:

$$B_{\text{count}} = \sum_{i=1}^{i=m} \frac{y_i \log(x_{(2m-i+1)}) - y_{(2m-i+1)} \log(x_i)}{\log(x_{(2m-i+1)}) - \log(x_i)} \quad B_1 = \frac{B_{\text{count}}}{m} \quad (4-15)$$

$$A_{\text{count}} = \sum_{i=1}^{i=m} \frac{y_i - B_1}{\log(x_i)} \quad A_1 = \frac{A_{\text{count}}}{m} \quad (4-16)$$

For the second part of this piecewise function, the linear expression  $A_2 x + B_2$  is simply calculated following a similar process than logarithmic expression.

$$A_2 = \frac{y_m - y_j}{x_m - x_j} \quad (4-17)$$

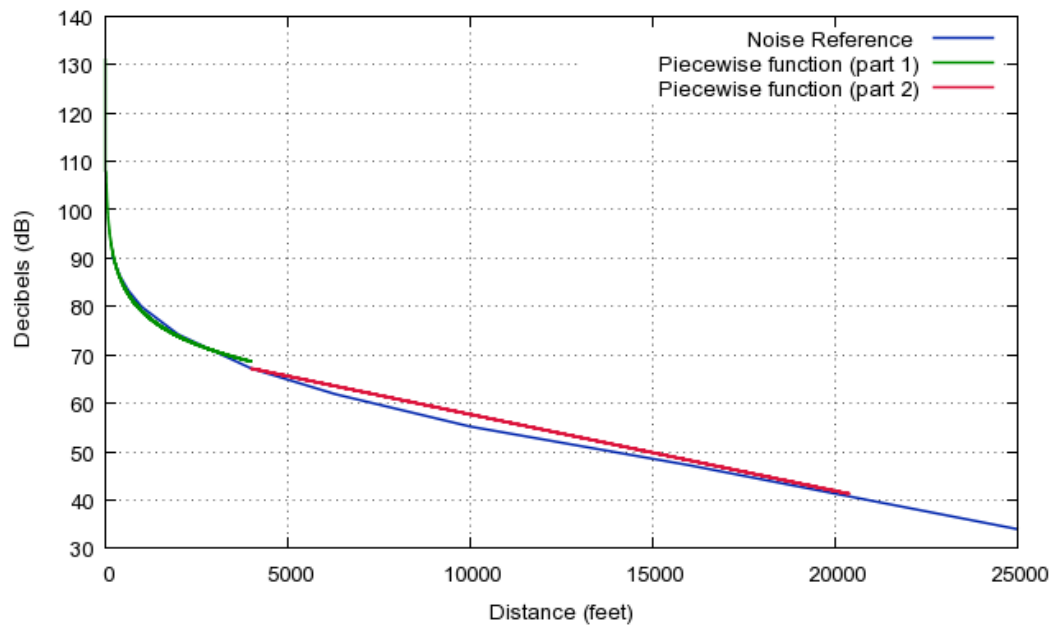
$$B_2 = y_j \quad (4-18)$$

Where  $j$  refers to position where  $x = 4000\text{ft}$  and piecewise function changes to linear approximation. MATLAB® code for this particular case can be found in Appendix E.

The result of applying this approximation method, is a piecewise function represented in Figure 4-2. First part of this function (green) uses a logarithm approximation, while second part is a linear approximation (red).

Table 4-1 shows a comparison between real values of sound exposure level of an engine similar to Pratt & Whitney PW-2036 versus approximated values using piecewise function. It is visible that maximum approximation error obtained is -

2.5 dB which is an acceptable value considering the distance of the observer (10,000 ft).



**Figure 4-2:** PW2036-like noise data approximation

**Table 4-1:** Comparison between empirical noise data and approximated data

Distance (ft)	Empirical Value (dB)	Piecewise function (dB)	Error (dB)	Error (%)
200	90.2	91.2	-1	1.10
400	86.3	85.9	0.4	0.46
630	83.3	82.5	0.8	0.96
1,000	79.9	79.1	0.8	1.00
2,000	74.2	73.8	0.4	0.53
4,000	67.2	67.2	0	0
6,300	61.9	63.5	-1.6	2.58
10,000	55.2	57.7	-2.5	4.52

<b>16,000</b>	47.2	48.2	-1	2.11
<b>25,000</b>	34.0	34.0	0	0

Equations described in (4-15) is used to compute Sound Exposure Level (SEL) relative to two (2) observers.

The 4DT generator has been designed to reduce noise during take-off and departure phases in a particular area. The main idea behind the use of *observers* is that they should be placed in the edge of populated areas or villages affected by noise so that flying close to them is avoided.

Optimization process tries to find a solution so that noise levels at these *observers* are minimal.

SEL value is used to calculate a Noise performance index which aims at minimize noise while following the target altitude:

$$J_{noise}(t) = \int_{t_0}^{t_f} \left( (K_1 L_{E1}(t) + K_2 L_{E2}(t))^2 + \frac{(z(t) - z_p)^2}{z_{max}} \right) dt \quad (4-19)$$

Where  $L_{E1}$  and  $L_{E2}$  denote the SEL relative to observers 1 and 2. Also  $K_1$  and  $K_2$  are big-enough scaling factors that are selected depending of the level of importance of each observer with respect to its geographic location (e.g. one of the observers could be more important, if this is closer to a populated area than other).

#### 4.4 Boundaries and Constraints

The boundaries and constraints used for defining the optimal control problem are defined by aircraft performance limits and flight plan. The states and control variable limits are defined by:

**Table 4-2:** Control variable constraints

Control variables
$\alpha_{min} < \alpha(t) < \alpha_{max}$
$\phi_{min} < \phi(t) < \phi_{max}$
$T_{min} < T(t) < T_{max}$

**Table 4-3:** States variables constraints

State variables
$x_{min} < x(t) < x_{max}$
$y_{min} < y(t) < y_{max}$
$z_{min} < z(t) < z_{max}$
$V_{min} < V(t) < V_{max}$
$\psi_{min} < \psi(t) < \psi_{max}$
$\gamma_{min} < \gamma(t) < \gamma_{max}$
$m_{min} < m(t) < m_{max}$

Constraints for variables  $x$  and  $y$  are defined by flight plan segments. Each segment is composed by two waypoints, therefore the limits of  $x$  and  $y$  variables of the segment are defined within the initial and final (target) waypoint.

Additionally, the heading ( $\psi$ ) constraint is defined by minimum value 0 degree and maximum value 359 degrees.

**Table 4-4:** Default constraints

Variable	Maximum Value	Minimum value
East distance ( $x$ )	Target point east distance	Previous point east distance
North distance( $y$ )	Target point north distance	Previous point north distance
Heading ( $\psi$ )	0°	359°

Other constraints are different for each particular case and cannot be specified a particular interval because they depend of aircraft performance limits.

Each aircraft has different performance limits, for this reason, the optimal control problem has been designed to obtain this information from an aircraft settings text file.

An example of the constraints for a C-17 *Globemaster III* aircraft, containing upper and lower boundaries for altitude ( $z$ ), velocity ( $V$ ), path angle ( $\gamma$ ), bank angle ( $\phi$ ), angle of attack ( $\alpha$ ), thrust ( $T$ ) and aircraft mass ( $m$ ) is shown in Figure 4-3.

```

* ----- *
* 4DT Generator v1.0a | Aircraft Settings File *
* Developed by Manuel Amaro (m.a.amarocarmona@cranfield.ac.uk) *
* Cranfield University, United Kingdom | 2015 *
* ----- *

AIRCRAFT_MODEL      = C-17 Globemaster III
ENGINE_MODEL         = PW2000

----- AIRCRAFT LIMITS -----
MIN_SPD              = 140
MAX_SPD              = 400
MIN_PATH_ANGLE       = -5
MAX_PATH_ANGLE       = 5
MIN_BANK_ANGLE       = -15
MAX_BANK_ANGLE       = 15
MIN_ATTACK_ANGLE     = -5
MAX_ATTACK_ANGLE     = 5
MIN_THRUST           = 0
MAX_THRUST           = 161200
MIN_MASS             = 185000
MAX_MASS             = 585000
MAX_ALTITUDE         = 40000

```

**Figure 4-3:** Aircraft settings example

Speed values are expressed in knots, angle values in angles, thrust in pounds (force), mass in pounds and altitude values in feet.

Additionally to states variables, initial and final time constraints are described based on following equations:

$$t_{p+1 (min)} = t_{p (min)} + \frac{d_p}{V_{max}} \quad (4-20)$$

$$t_{p+1 (max)} = t_{p (max)} + \frac{d_p}{V_{min}} \quad (4-21)$$

Where  $t_{p+1}$  and  $t_p$  is time at next and previous segment respectively,  $V_{max}$  and  $V_{min}$  is aircraft maximum and minimum velocity respectively; finally  $d$  is defined as the orthodomic distance from initial to target waypoint  $p$ .

#### 4.5 Overview of 4D Trajectories Research Software (4DT RS)

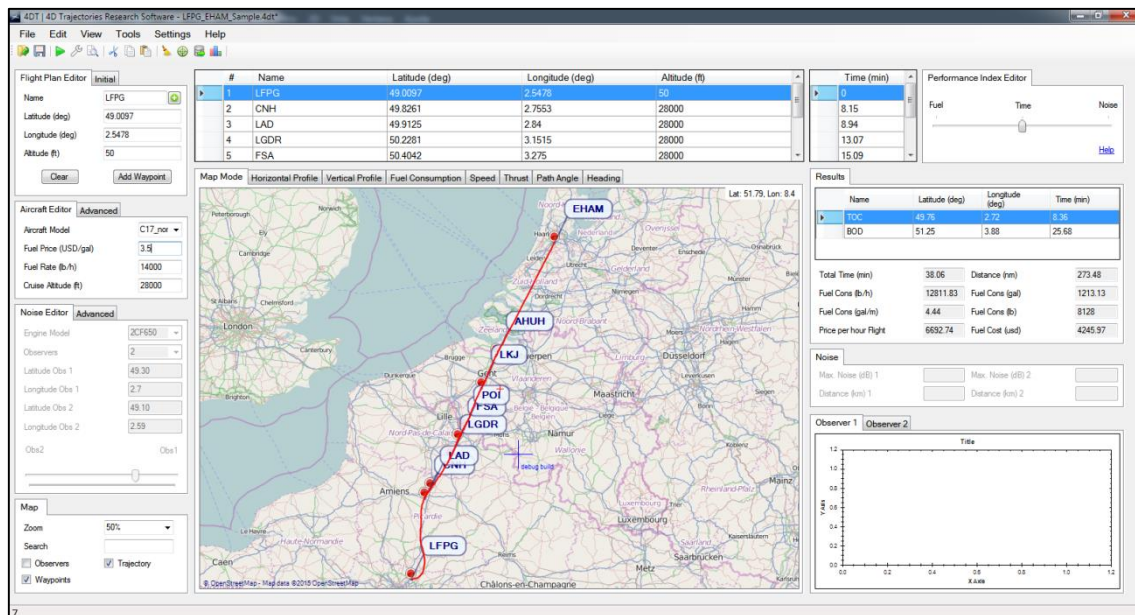
4D Trajectories Research Software Suite (4DT RS) is a software package that has been designed and implemented based on the 4DT generator. The software has been designed to generate, analyse and test optimal trajectories in order to reduce fuel, time and noise.

Trajectories are created based on a given flight plan that must be composed by waypoints.

4D Trajectories Research Software includes a user-friendly graphic interface and an installer for Microsoft® Windows® 7 or superior. User interface is shown in Figure 4-4.

4DT RS uses PSOPT [70] as internal optimal control solver which uses pseudo-spectral methods to solve the problem.





**Figure 4-4: 4DT RS Graphic User Interface**

Once application starts, the user could select whether loading a previously defined case or building a new case by creating / importing a flight plan. For creating flight plans it has been included a waypoints database which contains more than 150,000 airports, fixes, NDB and VOR stations. Additionally, initial conditions, aircraft model, and typical optimal control problem are input parameters that can be added by the user.

Generated trajectories parameters are represented in a map, vertical and horizontal profiles are shown in plots. Optionally, trajectory data can be exported as data files to be used in other software.

4DT RS has been provided with a Tracking & Guidance module linked to a commercial flight simulator using an UDP connection. This module is used to simulate a guidance system as well as testing and validating the generated trajectories. Tracking & guidance module is described in Chapter 5.

Additionally, 4DT RS provides the user with more utilities such as: *detailed view* tool, importing & exporting flight plans, noise levels contours and trajectories comparison capabilities.

The following sections describe the structure and components of this software.

## 4.6 Software Structure

4DT RS structure is mainly composed by a set of algorithms that setup a generic optimal control problem and generate a 4D optimal trajectory. These algorithms have been assembled into one single module responsible for building and solving the optimal control problem. This module is known as *4D Trajectories Generator Core* (4DT RS Core) and basically includes all the concepts boarded in sections 4.2 and 4.3.

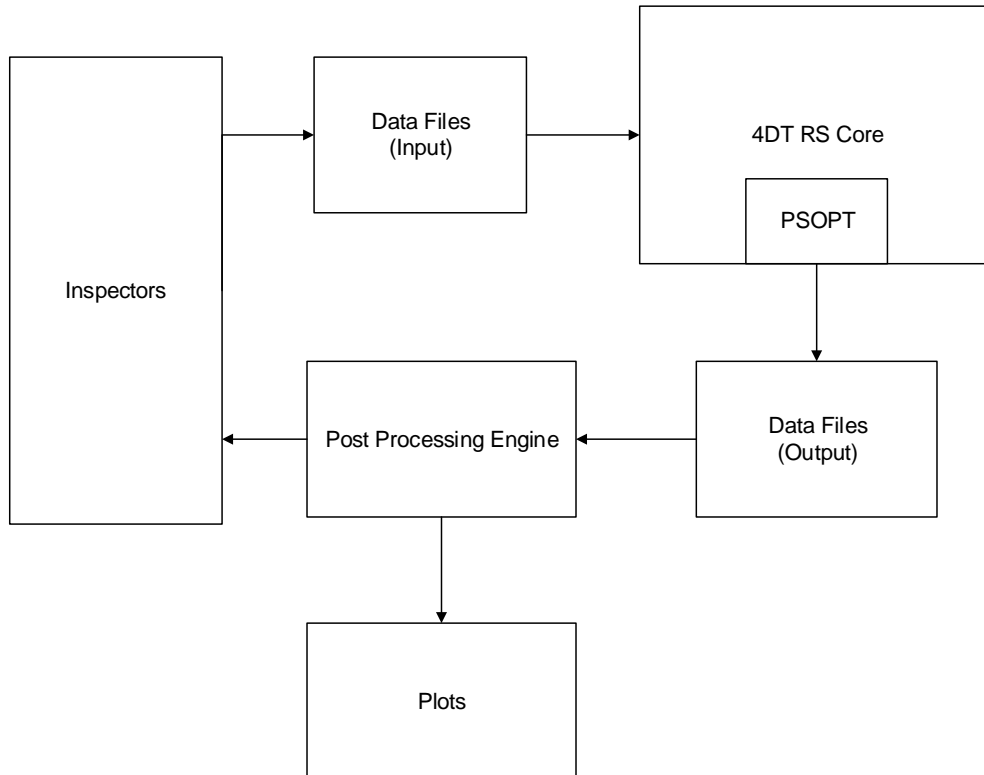
Additionally to 4DT RS Core, it has been implemented a Graphic User Interface (GUI) that contains the functions used to input data into 4DT RS Core and also includes tools that compute flight data relative to optimal trajectories synthesized and it is useful to evaluate and compare the optimal trajectories

Figure 4-5 shows a top-level overview of 4DT RS structure. A full view of 4DT RS structure is available in Appendix A.

The block located at top-right represents the 4DT RS Core. This core receives data via text files from data input modules named *Inspectors*. Each inspector provides a different type of data relative to the optimal trajectory problem. For example, Flight Plan Inspector provides inputs relative to trajectory flight plan such as waypoints latitude, longitude, name or altitude. Similarly, aircraft inspector provides inputs relative to aircraft performance model and noise inspector inputs data relative to noise computation.

Other data is computed directly inside the 4DT RS Core (e.g. OCP time limits or aircraft dynamics model). Elements relative to inputs (Inspectors) are represented by blocks located at left side of Figure 4-5.

4DT RS Core solution is exported via text files and data is processed to be shown in the graphic user interface via Inspectors and Plots.



**Figure 4-5: 4D Trajectories Research Software Structure**

## 4.7 4D Trajectories Generator Core

The 4DT Generator Core is a C++ executable stand-alone application that uses a general flight plan in order to generate predicted aircraft optimal trajectories focused on reducing a performance index specification. This application is considered the nucleus of the 4DT Research Software and it has been compiled and tested on GNU Linux Ubuntu 14 and Microsoft® Windows® 7.

The 4DT generator core has been designed and implemented as an application that setup an optimal control problem using the information provided by the user.

Subsequently, the optimal control problem is solved by PSOPT [70]. Pseudo-spectral discretization of time-dependent variables is performed using Legendre method (by default). The resultant NLP problem is solved using IPOPT (alternatively, SNOPT [71] non-linear programming solver can be used).

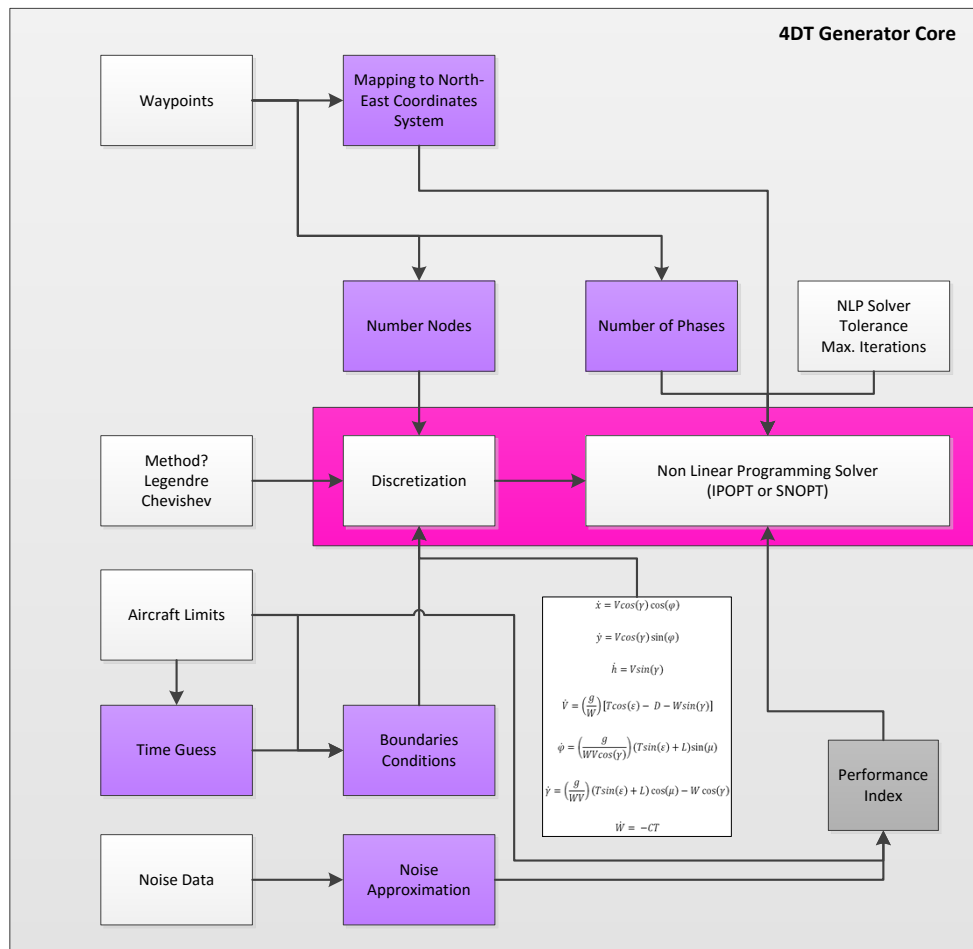
Figure 4-6 shows 4DT RS Core structure. It is composed by elements described in previous sections: aircraft dynamics mathematical model (4.2), performance

index specification (4.3), set of boundaries/constraints (4.4), Optimal Control Solver functions (PSOPT libraries), and Nonlinear programming solver (IPOPT).

4DT RS Core receives four (4) main inputs:

- 1) A set of waypoints (flight plan)
- 2) A set of states and control constraints (aircraft performance limits)
- 3) Noise data specification (empirical data to calculate noise)
- 4) A set of PSOPT setup variables (discretization method, tolerance...)

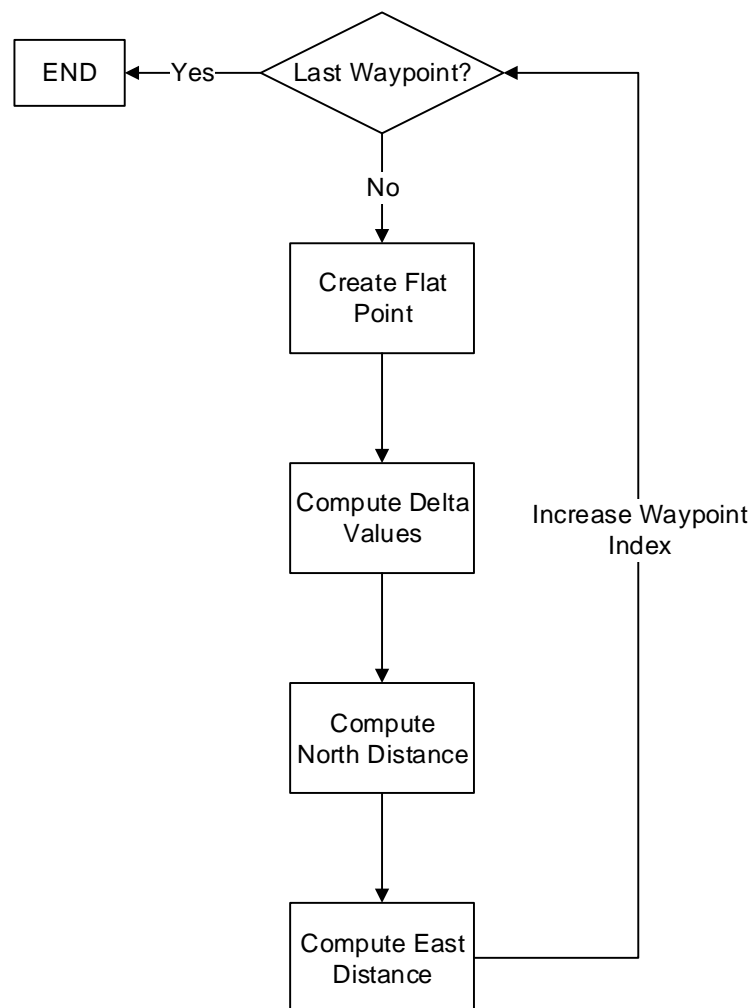
Initially, flight plan is decomposed in different segments in order to treat the problem as a *multiphase optimal control problem* so that each phase is referenced to one *segment*. Each phase is linked to previous phase so that total solution results as optimal for the whole trajectory. This is achieved by making use of internal PSOPT linkages functions (ref. PSOPT Manual [70]).



**Figure 4-6: 4DT RS Core internal structure**

Waypoints are mapped from geographical coordinates to a north-east coordinates (flat point) system by using a World Geodetic System (WGS)-based function that basically converts units from degrees to standard longitude units by mapping each waypoint with respect to a reference point.

Figure 4-7 shows the mapping function algorithm. Also, a detailed C# version of this function is available in Appendix C.



**Figure 4-7:** MappingWGS function algorithm

Returned value by *MappingWGS()* function is a *flat\_point* structure which is described in the following table:

**Table 4-5: *flat\_point* struct**

Value	Type	Description
north_dist	double	Y distance from reference point to waypoint
east_dist	double	X distance from reference point to waypoint

Additionally, the number of waypoints is used to obtain the number of phases that contains the optimal control problem. Each segment composed by initial waypoint and final waypoint is processed as one problem phase.

Aircraft limits compose most of the constraints used to describe the optimal control problem. Values are imported from the aircraft performance limits file (shown previously in Figure 4-3), processed and included into PSOPT solution by using the *upper and lower bounds* structures (similarly are included the initial conditions).

Following table shows a typical PSOPT syntaxes of states and control boundaries structures.

**Table 4-6: PSOPT boundaries variable syntaxes in C++**

```
// Lower bounds (states)
problem.phases(i).bounds.lower.states(1)    = lon_lower[i-1];
problem.phases(i).bounds.lower.states(2)    = lat_lower[i-1];
problem.phases(i).bounds.lower.states(3)    = alt_lower[i-1];
problem.phases(i).bounds.lower.states(4)    = velocity_lower[i-1];
problem.phases(i).bounds.lower.states(5)    = heading_lower[i-1];
problem.phases(i).bounds.lower.states(6)    = path_angle_lower[i-1];
problem.phases(i).bounds.lower.states(7)    = mass_lower[i-1];

// Upper bounds (states)
problem.phases(i).bounds.upper.states(1)    = lon_upper[i-1];
problem.phases(i).bounds.upper.states(2)    = lat_upper[i-1];
problem.phases(i).bounds.upper.states(3)    = alt_upper[i-1];
problem.phases(i).bounds.upper.states(4)    = velocity_upper[i-1];
problem.phases(i).bounds.upper.states(5)    = heading_upper[i-1];
problem.phases(i).bounds.upper.states(6)    = path_angle_upper[i-1];
problem.phases(i).bounds.upper.states(7)    = mass_upper[i-1];

// Lower bounds (control)
problem.phases(i).bounds.lower.controls(2)  = bank_angle_lower[i-1];
```

```

problem.phases(i).bounds.lower.controls(1) = angle_attack_lower[i-1];
problem.phases(i).bounds.lower.controls(3) = thrust_lower[i-1];

// Upper bounds (control)
problem.phases(i).bounds.upper.controls(2) = bank_angle_upper[i-1];
problem.phases(i).bounds.upper.controls(1) = angle_attack_upper[i-1];
problem.phases(i).bounds.upper.controls(3) = thrust_upper[i-1];

```

Additionally, these limits are used by time guess function in order to compute maximum and minimum time constraints by using the equations (4-20) and (4-21).

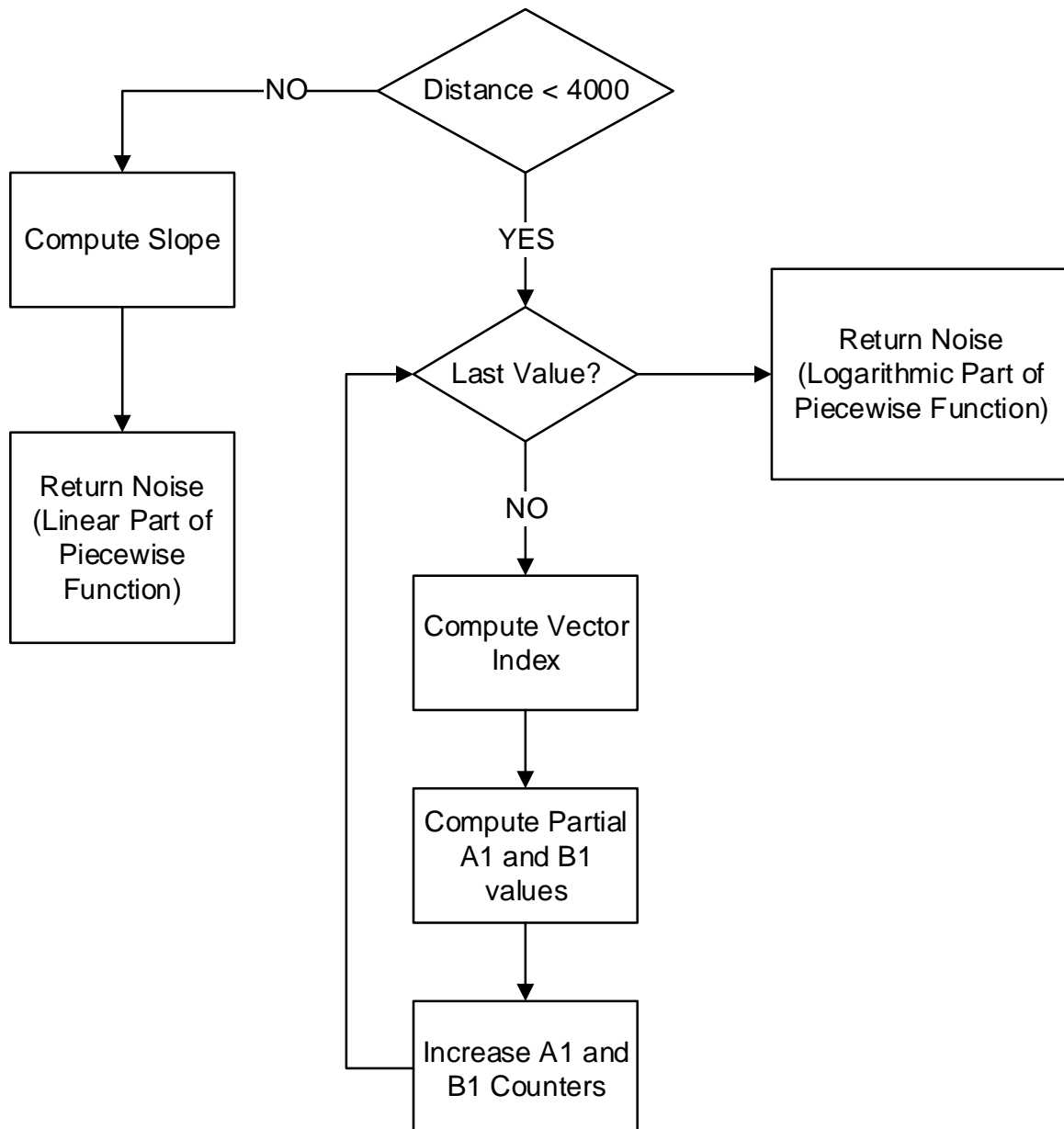
Noise data is received and converted into a vector of *noise\_data* objects. Each object *noise\_data* is composed by the following information:

**Table 4-7: *noise\_data* struct**

Value	Type	Description
OP	string	Operation (departure by default)
MET	string	Type of noise (SEL by default)
ENGINE	string	Engine Model
noise200	double	Noise perceived at 200 ft
noise400	double	Noise perceived at 400 ft
noise630	double	Noise perceived at 630 ft
noise1000	double	Noise perceived at 1,000 ft
noise2000	double	Noise perceived at 2,000 ft
noise4000	double	Noise perceived at 4,000 ft
noise 6300	double	Noise perceived at 6,300 ft
noise10000	double	Noise perceived at 10,000 ft

noise20000	double	Noise perceived at 20,000 ft
noise25000	double	Noise perceived at 25,000 ft

This data is used to compute Sound Exposure Levels (SEL) by using the piecewise function described in equation (4-13) and shown in the following figure:

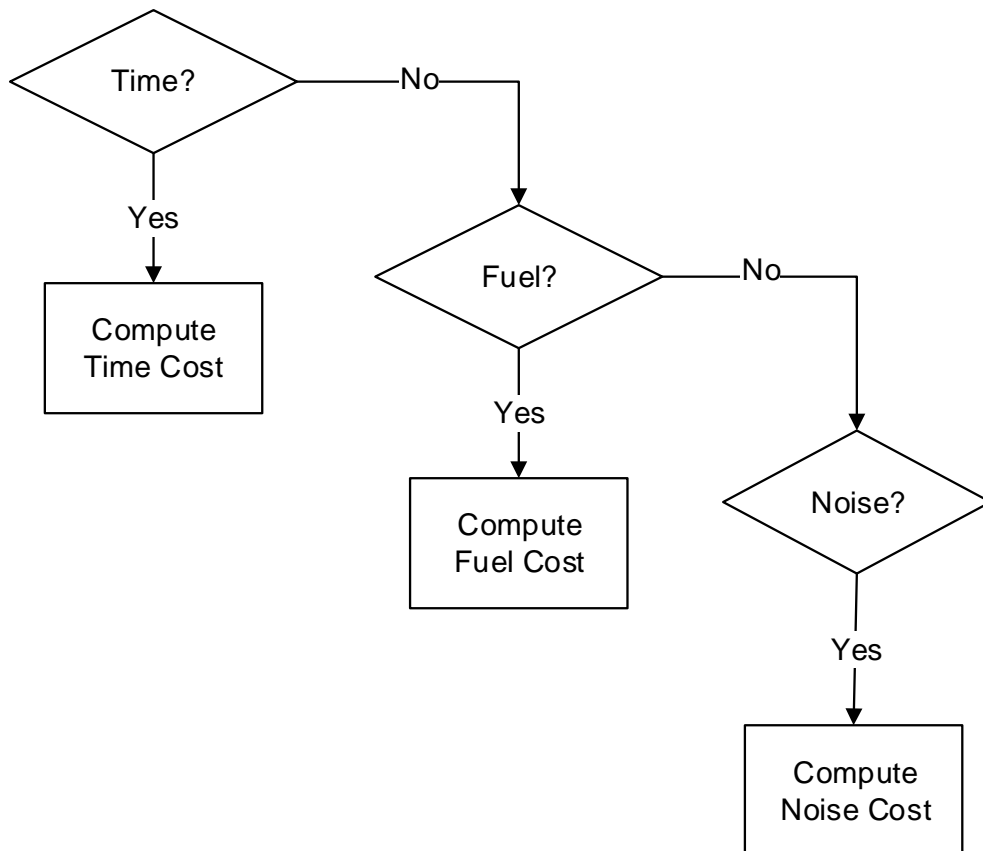


**Figure 4-8:** Compute\_noise function algorithm



The performance specifications are calculated in a function named *integrand\_cost()*. This function contains the performance index specifications for fuel, time and noise described in section 4.3 and basically selects one of them, depending of user choice.

Figure 4-9 shows the decision algorithm implemented in *integrand\_cost()* function.



**Figure 4-9:** Integrand\_cost function algorithm

PSOPT requires some configuration parameters such as discretization method, nonlinear programming solver, tolerance and maximum number of iterations. These parameters are read from a general settings file and input into a *input\_settings* struct, which is an object that contains general configuration parameters for PSOPT and 4DT RS Core.

**Table 4-8:** Example of PSOPT setting variables sintaxis in C++

```
algorithm.nlp_iter_max      = input_settings.max_iter;
algorithm.nlp_tolerance    = input_settings.tolerance;
algorithm.nlp_method       = input_settings.nlp;
algorithm.scaling          = "automatic";
algorithm.derivatives      = "automatic";
algorithm.print_level      = 1;
```

Once optimal control problem is formulated, it is called the function that executes PSOPT: *psopt(solution, problem, algorithm)* in order to find a solution to the problem.

#### 4.7.1 Exporting Results and Noise Grid

If optimal solution is found, a new trajectory is defined by the optimal values of states and control variables at each collocation point (node). The total number of values is equivalent to *number of nodes x number of flight plan segments*.

These variables are exported independently as files that are used to represent the trajectory by the Graphic User Interface (GUI).

For each state and control variable it is exported one text file with the name of that variable (e.g altitude.dat, heading.dat, thrust.dat). A set of optimal trajectory latitude values output file example is shown in Figure 4-10.

49.0097	49.0097	49.0097	49.0097	49.0098
49.0103	49.0117	49.0148	49.0205	49.0298
49.0436	49.0635	49.091	49.1276	49.174
49.2294	49.292	49.3591	49.4288	49.4989
49.568	49.6346	49.6981	49.7576	49.8132
49.8649	49.913	49.9579	49.9997	50.0386
50.0743	50.1066	50.1352	50.1598	50.1803
50.197	50.21	50.2195	50.2255	50.2281

**Figure 4-10:** Example of data output by 4DT RS Core

For PSOPT particular case, data regarding to each iteration is exported independently in a text file. This data is available to the user via an integrated built-in text viewer.

In order to produce noise level contours, it is calculated the maximum Sound Exposure Level (SEL) perceived at each point of a terminal/departure area.

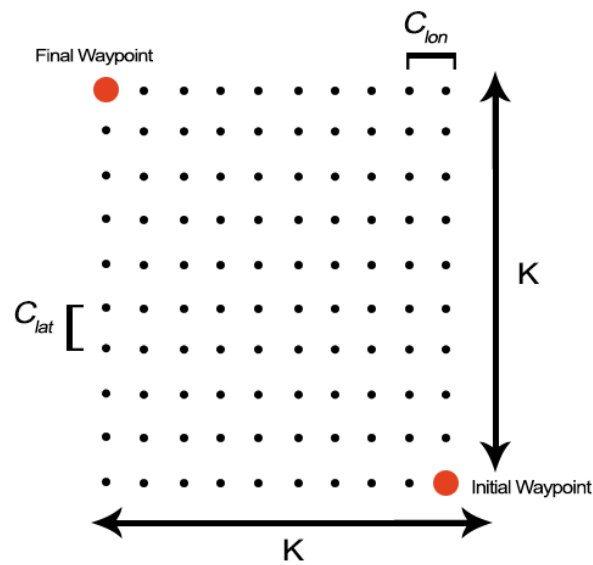
A grid is defined by points separated by distances  $C_{lat}$  in Y axis and  $C_{lon}$  in X axis:

$$C_{lat} = \left| \frac{Lat_i - Lat_f}{K} \right| \quad (4-22)$$

$$C_{lon} = \left| \frac{Lon_i - Lon_f}{K} \right| \quad (4-23)$$

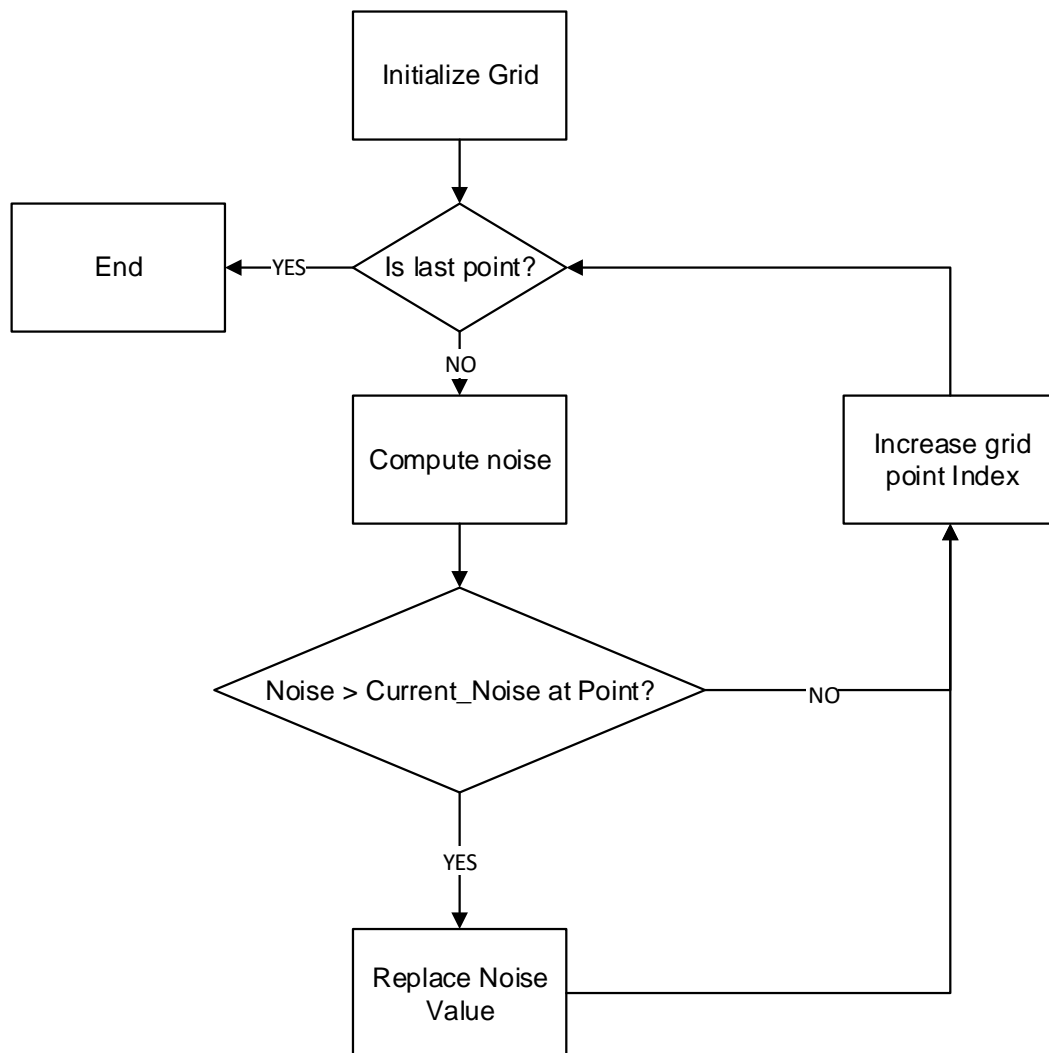
where  $K$  is a value which defines the grid size. Since noise optimization is carried out for departure phases which usually involve areas with limited size, it has been determined that  $K = 20$  provides an acceptable balance between noise grid resolution and processing CPU time.

Figure 4-11 shows the matrix of elements exported as noise grid.



**Figure 4-11: Noise Grid**

The noise is computed for each point of the grid and the distance is calculated respect to aircraft trajectory. Since aircraft position varies along the trajectory, SEL value is calculated for each grid point as many as aircraft position entries are available in trajectory files output by PSOPT (*number of nodes x number of phases*).



**Figure 4-12:** Noise Grid Computation algorithm

A resultant vector containing grid points and maximum SEL value perceived is obtained. For noise computation, it is obviously used the same *compute\_noise()* function used for noise performance index specification (equation (4-13)).

## 4.8 Graphic User Interface (GUI)

A graphic user interface includes a set of components that make data input process more user-friendly. Also it contains functions and elements that are used for comparing, representing and evaluating the resultant trajectory data. This section describes the different components included in GUI and shows important aspects of structure and tools of 4DT RS.

### 4.8.1 Maps, Waypoints and Data validation

Before running the optimization, flight plan is represented by waypoints in a built-in 2D map that is rendered by *GreatMaps* libraries [64].

*GreatMaps* is a set of C# libraries for .NET framework [68]. These libraries includes a set of functions that can be used to create, edit and interact with two-dimensional maps.

The base object of these libraries is the object *PointLatLng*

**Table 4-9:** *PointLatLng* struct

Value	Type	Description
Latitude	double	Latitude in degree
Longitude	double	Longitude in degree

In case of representing a *PointLatLng* object in a map, it is necessary to create a map marker. A map marker is represented by objects *GMapMarker*. Basically this object links the latitude and longitude position to its relevant position over an image. For this reason, this object is constructed by a *PointLatLng* element and a *Bitmap* image that represent the marker icon.

**Table 4-10:** *GMapMarker* struct

Value	Type	Description
Point	PointLatLng	Point to be represented
Icon	Bitmap	Icon to be rendered

In case of representing a line in a map, this is performed by making use of the object *GMapRoute* that basically receives a list of *PointLatLng* elements.

**Table 4-11:** *GMapRoute* struct

Value	Type	Description
Route	List<PointLatLng>	List of points that create a path
Name	String	Name of the route

*GMapMarker* or *GMapRoute* objects are grouped by layers of type *GMapOverlay*. These layers can group several markers, routes or other objects such as polygons over a map. For this reason it has been created a *GMapOverlay* layers for waypoints, trajectories, observers and polygons.

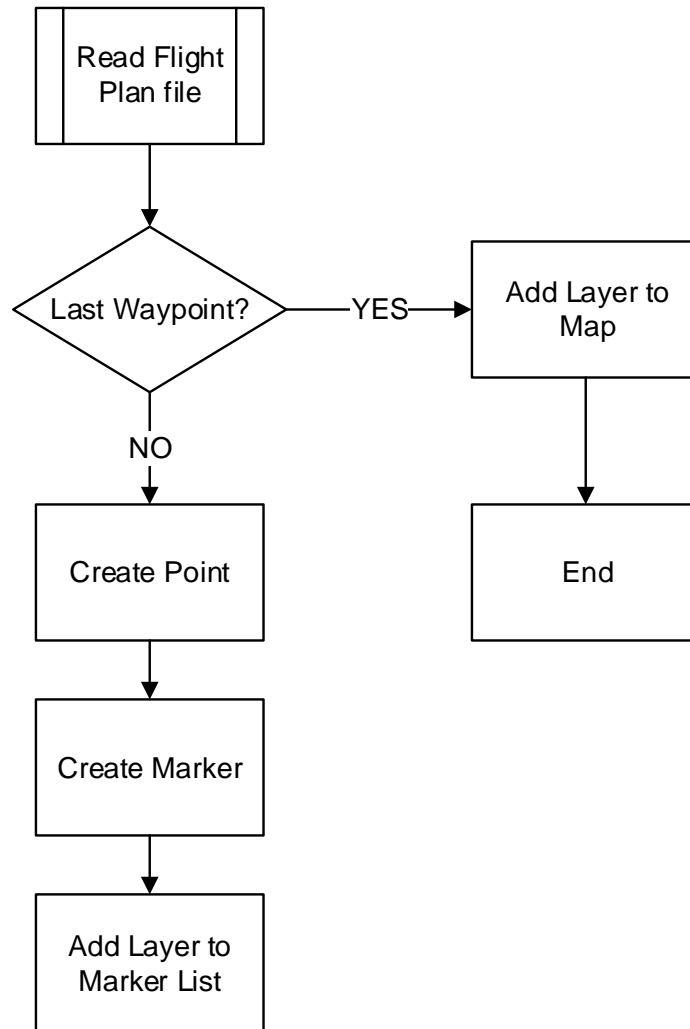
Finally, each layer of type *GMapOverlay* is linked to a *GMapControl* which is the graphic element linked to a map provider and is placed into the GUI main form. There are few map providers, for this software it has been used OpenStreetMaps.org which is a free map provider.

In summary the main objects used from *GreatMaps* libraries for this software are:

**Table 4-12:** Objects and elements used

Element	Object	Description
Point	<i>PointLatLng</i>	Point which contains geographical information data
Line	<i>GMapRoute</i>	List of points which contains a path or route
Marker	<i>GMapMarker</i>	Object that associates a <i>PointLatLng</i> to an image
Layer	<i>GMapOverlay</i>	Object that groups several objects <i>GMapMarkers</i>
Map	<i>GMapControl</i>	A graphic element linked to a map provider.

For representing waypoints, it has been created the function *PlotWaypoints()* that basically receives data of one or more waypoints from a text file and plots the waypoints into a *GMapControl* element. This is achieved by creating a point, creating a marker, adding it to a layer and finally adding the layer to the map.



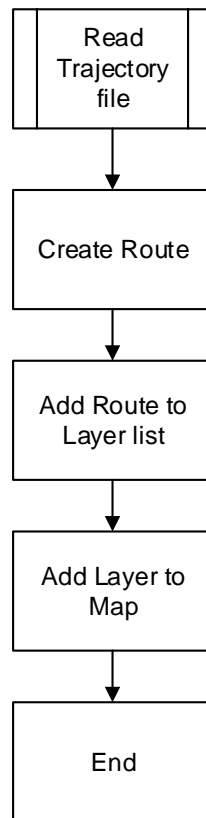
**Figure 4-13:** PlotWaypoints function algorithm

*GMapMarker* elements are also provided with different components that can be adjusted in order to obtain different rendering effects. For markers, it is possible to add text and to adjust how the information is shown. Adjusting properly all these elements contributes to obtain the user-friendly interface of 4DT RS.

For representing trajectories, it has been created the function *PlotTrajectory()*, which basically reads the trajectory files exported by 4DT RS Core (*lat.dat* and *lon.dat*), creates an object *GMapRoute* and adds this object to the relevant layer, which is then plotted on the map.

Similar than markers, *GMapRoute* visual properties are adjusted in order to change line colours and stroke. Thinking on possible comparison between

optimal trajectories, *PlotTrajectory()* function includes a routine that changes line colour property each time it is executed (includes a limit of 3 times). This routine has been hidden in the function definition of this document to simplify the function overview.



**Figure 4-14:** PlotTrajectory function algorithm

Additionally to *GreatMaps* elements described, some *WindowsForms* elements contained in .NET Framework has been used to create the user graphic interface. Since the extensive amount of components contained in *WindowsForms* .NET class, more information of control and objects of this framework can be found at [68].

However, a summary of most used elements to represent 4DT RS GUI is shown in the following table:



**Table 4-13:** WindowsForms Controls

Element	Object	Description
Table	<i>DataGridView</i>	Control used to represent tables
Text Field	<i>TextBox</i>	Control used to represent text field
Button	<i>Button</i>	Control used to represent button
Tabs	<i>TabControl</i>	Control used to sort information in tabs
Select	<i>ComboBox</i>	Control used to create selection boxes
Track Bar	<i>TrackBar</i>	Control used to show trackbar
Toolbar Icon	<i>ToolStripButton</i>	Control used to create icons in the toolbar
Toolbar Item	<i>ToolStripMenu</i>	Control used to create entries in menu

Input data for waypoints, observers and initial conditions is provided by the user via *TextBox* controls. Data validation is carried out to provide consistent values of latitude, longitude, altitude, and heading, as well as to guarantee that no waypoint is repeated. Validation is carried out by the following set of functions:

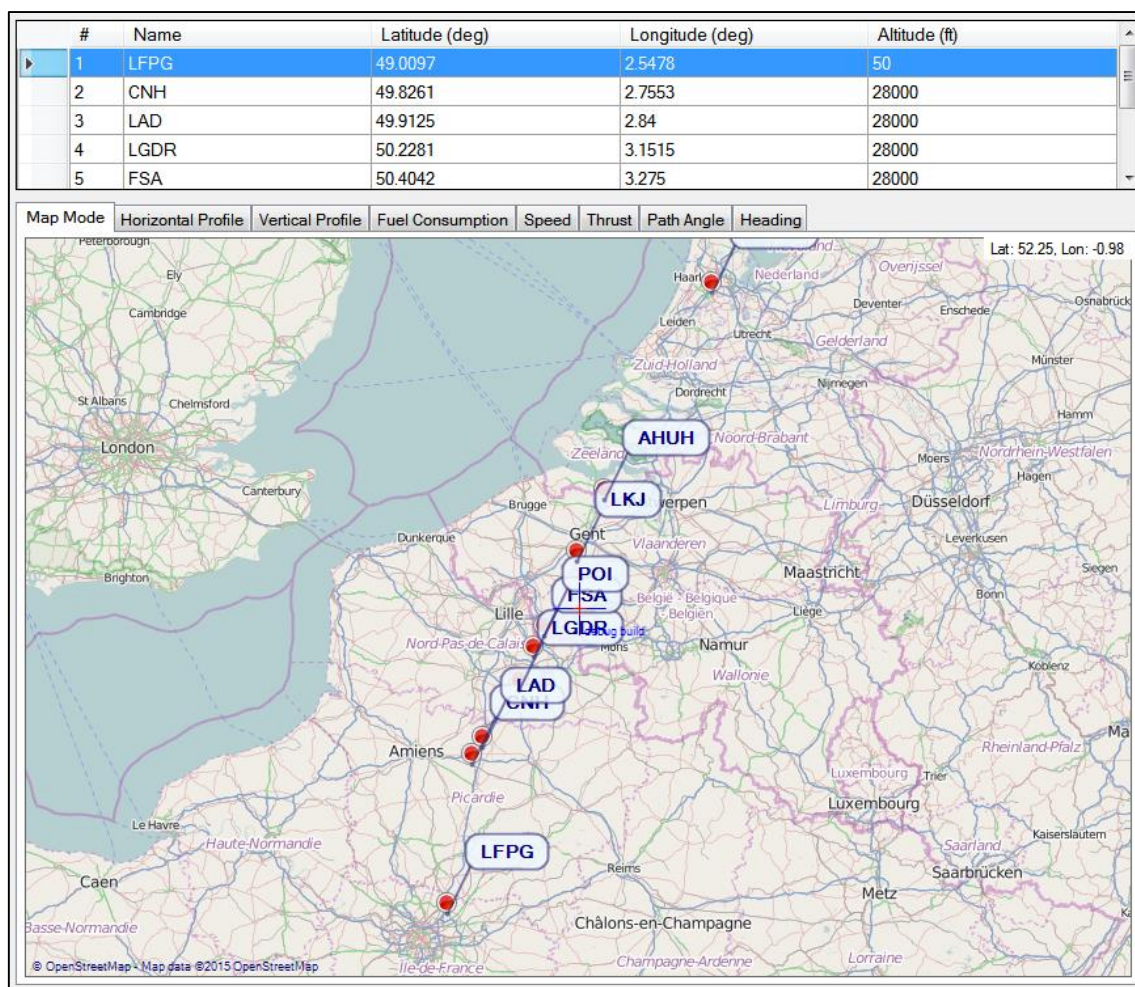
**Table 4-14:** Validation functions

Function	Description
<i>IsHeading</i>	Returns true if input data is in the interval [0, 360]
<i>IsLatitude</i>	Returns true if input data is in the interval [-90, 90]
<i>IsLongitude</i>	Returns true if input data is in the interval [-180, 180]
<i>IsValidName</i>	Returns true if input is a no-repeated alphanumeric
<i>IsPositiveNumeric</i>	Returns true if input data is a positive number
<i>IsNumeric</i>	Returns true if input data is a number
<i>TypeCheck</i>	Returns true if a string is "TIME", "FUEL" or "NOISE"

Once data is validated, it is copied from the *TextBox* controls to *Flight Plan table*, which is represented by a *DataGridView* control. Once a new entry is added to Flight Plan table, it is called the function *PlotWaypoints()* to plot the waypoints.

It is guaranteed that every time the user inputs data into the Flight Plan table, the waypoint is represented into the map, including a *tooltip* to show the name of the waypoint.

This idea is shown in the following Figure 4-15. It is visible an example of the Flight Plan table (*DataGridView* control), the map (*GMapControl*) and some waypoint markers (*GMapMarkers*).



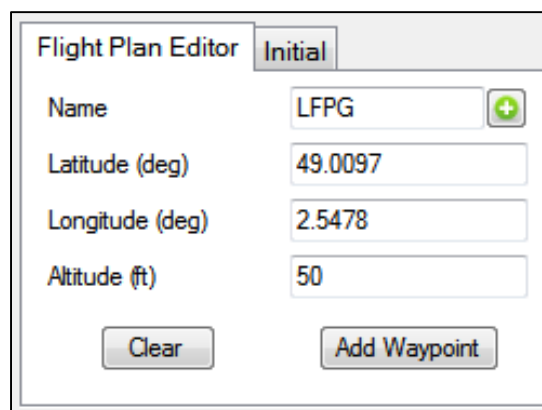
**Figure 4-15: Map and Flight Plan Table**

In this section, it has been introduced the idea that data is input by *TextBox* fields. These textbox fields are used to input different type of data such as flight plan, noise or aircraft performance data. In order to design a graphic interface thinking on an experience as much as user-friendly as possible, most of *TextBox* controls have been grouped into elements named *Inspectors*. There are different *Inspectors* depending of the type of data they are meant to input to the 4D Generator. In the following sections, it is described the available inspectors.

#### 4.8.2 Flight Plan Inspector

From the user point-of-view, the flight plan is the most important input. A flight plan defines the waypoints that aircraft has to fly-by/over before arriving to final destination. Flight plans for flight management systems are defined in ARINC 424 standard [65], [58]. A simplified version of this standard has been implemented in 4DT RS where each waypoint is composed by three dimensional points in the space defined by latitude, longitude and altitude.

The main idea of this editor is to provide to user with a friendly and responsive way to create a flight plan. Data is also reproduced for visualization into a table.

The image shows a software window titled "Flight Plan Editor" with a tab labeled "Initial". Inside the window, there are four text input fields arranged vertically. The first field is labeled "Name" and contains the text "LFPG". To the right of this field is a small green square button with a white plus sign. The second field is labeled "Latitude (deg)" and contains the value "49.0097". The third field is labeled "Longitude (deg)" and contains the value "2.5478". The fourth field is labeled "Altitude (ft)" and contains the value "50". At the bottom of the window, there are two buttons: "Clear" on the left and "Add Waypoint" on the right.

**Figure 4-16:** Flight Plan Inspector

Once it has been defined proper values for latitude, longitude and altitude a new waypoint entry is added to *Flight Plan table*, located in the upper-middle part of the GUI. Additionally, a new point and tag is added to the map.

In addition, directly interaction between maps and editor fields can be achieved by using the mouse pointer. In this manner, if user clicks on map, the latitude and

longitude of its location is copied into editor relevant fields. Since map provides 2D information, altitude field has to be directly filled by user.

Despite of adding latitude and longitude values by clicking on the map is a very practical method, it is not considered quite accurate. For this reason, it has been added a navigation database where information regarding to real navigation data could be easily accessed.

ARINC 424 provides a standard format to represent flight plans on flight management systems. This database is composed by waypoints that represent different navigation elements mentioned in ARINC 424 standard. The database specification is shown in Table 4-15:

**Table 4-15:** Database specification

Table	Details	Entries
NDB	Non-directional beacon station (2-3 characters)	11,126
VOR	VHF Omni-directional range station (2-3 characters)	
Airport	Airport identified by ICAO code (4 characters)	44,684
FIX	FIX waypoint (5 characters)	119,721
		<b>175,531</b>

Accessing to database is performed using SQLite connection functions. A new object of *SQLiteConnection* is created with the file path of the database. The following table shows the functions used to connect to database connection.

**Table 4-16:** Functions used by *SQLiteConnection* and *cmd* objects

Function	Description
<i>SQLiteConnection</i>	Construct a new <i>SQLiteConnection</i> object
<i>Open</i>	Opens a database connection
<i>CreateCommand</i>	Creates a new <i>cmd</i> object that will be used to execute a database query

<i>CommandText</i>	Specifies a SQL query
<i>ExecuteReader</i>	Executes the SQL query provided by <i>CommandText</i>
<i>Close</i>	Closes a database connection

By using functions above, it has been implemented the function *SearchDatabase()* that opens a new database connection and creates a new *CommandText* object:

```
CommandText = "SELECT " + fields + " FROM " + table + extra;
```

This object contains a string that executes a different SQL query depending of the user choice. To understand this, note that *fields*, *table* and *extra* are string variables that can get different values depending of the table Airport, VOR-NDB or FIX. For this reason, these variables are previously configured according to the following table:

**Table 4-17:** Variable values for SQL queries

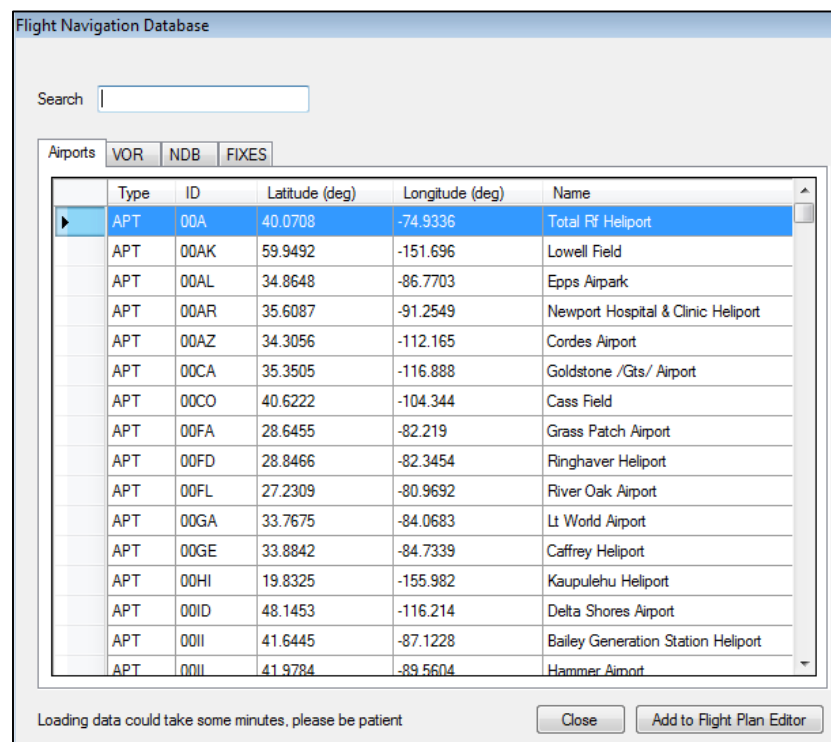
Type	Values
APT	table = "airports"; fields = "ident, latitude, longitude, name";
VOR	table = "navaids"; fields = "ident, latitude, longitude, name"; extra = " WHERE type = 'VOR-DME'";
NDB	table = "navaids"; fields = "ident, latitude, longitude, name"; extra = " WHERE type = 'NDB'";
FIX	table = "fixes"; fields = "ident, latitude, longitude";

Once data is retrieved from database, a new list is created and loaded into a *DataGridView* control. Finally the connection is closed.

From the user point-of-view, this data can be accessed by clicking on a plus (+) icon supplied in the flight plan editor. Information has been sorted in four different tabs: Airports, VOR, NDB and FIX.

Since the quantity of entries to be loaded into grid is considerable (total of 175,531), this process could take some time. For this reason, part of this data is loaded when application is executed and this process could delay loading forms and other application components. However, once data is loaded, it can be accessed directly by using the database viewer with no delay.

Figure 4-17 shows a typical representation of waypoints entries in navigation database form. Once selected, the waypoint information is added to flight plan editor fields and consequently it can be added to current flight plan table.



Type	ID	Latitude (deg)	Longitude (deg)	Name
APT	00A	40.0708	-74.9336	Total Rf Heliport
APT	00AK	59.9492	-151.696	Lowell Field
APT	00AL	34.8648	-86.7703	Epps Airpark
APT	00AR	35.6087	-91.2549	Newport Hospital & Clinic Heliport
APT	00AZ	34.3056	-112.165	Cordes Airport
APT	00CA	35.3505	-116.888	Goldstone /Gts/ Airport
APT	00CO	40.6222	-104.344	Cass Field
APT	00FA	28.6455	-82.219	Grass Patch Airport
APT	00FD	28.8466	-82.3454	Ringhaver Heliport
APT	00FL	27.2309	-80.9692	River Oak Airport
APT	00GA	33.7675	-84.0683	Lt World Airport
APT	00GE	33.8842	-84.7339	Caffrey Heliport
APT	00HI	19.8325	-155.982	Kaupulehu Heliport
APT	00ID	48.1453	-116.214	Delta Shores Airport
APT	00II	41.6445	-87.1228	Bailey Generation Station Heliport
APT	00II	41.9784	-89.5604	Hammer Airport

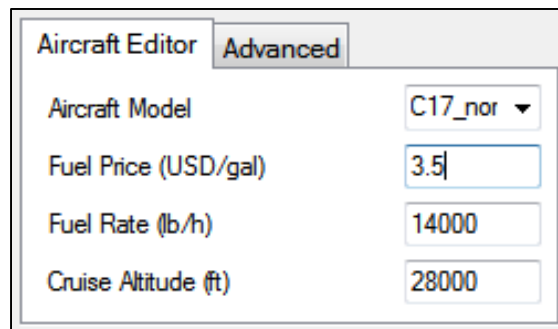
**Figure 4-17: Navigation Database**

Besides adding a set of waypoints using this editor, it is possible to export a defined flight plan. If this is the case, a text file contained relevant information about waypoints is generated. This file can be used to import the flight plan into 4DT RS in the future.

### 4.8.3 Aircraft and Initial Conditions Inspectors

Aircraft parameters used by 3DoF model are imported from aircraft settings. This includes aircraft performance limits, aerodynamics coefficients and fuel flow consumption coefficients.

Aircraft inspector is used to select aircraft models. Once loaded it shows all aircraft files contained in directory named “*Aircraft*” located in the 4DT RS root folder.



The screenshot shows a software window titled "Aircraft Editor" with a sub-tab labeled "Advanced". Inside the window, there are four input fields arranged vertically. The first field is "Aircraft Model" with a dropdown menu showing "C17\_nor". The second field is "Fuel Price (USD/gal)" with a text input containing "3.5". The third field is "Fuel Rate (lb/h)" with a text input containing "14000". The fourth field is "Cruise Altitude (ft)" with a text input containing "28000".

**Figure 4-18:** Aircraft and Initial Conditions Inspector

Additionally, fuel price (US dollars) and fuel consumption rate (lb/hour) are parameters requested to user. These parameters are used to calculate flight information once an optimal trajectory is found (described in section 4.10). In addition, cruise altitude is used to compute TOC and BOD points. 4DT RS has been provided with a built-in function that automatically fills this field in when importing a new flight plan composed by more than two (2) waypoints.

Initial conditions inspector is used to receive aircraft initial parameters. This is *initial speed*, commonly defined by aircraft  $V_2$  speed which depends of atmospheric conditions, weight and other parameters, *initial mass* which depends of loaded fuel into aircraft before departing and *initial heading* which depends of airport and runway.

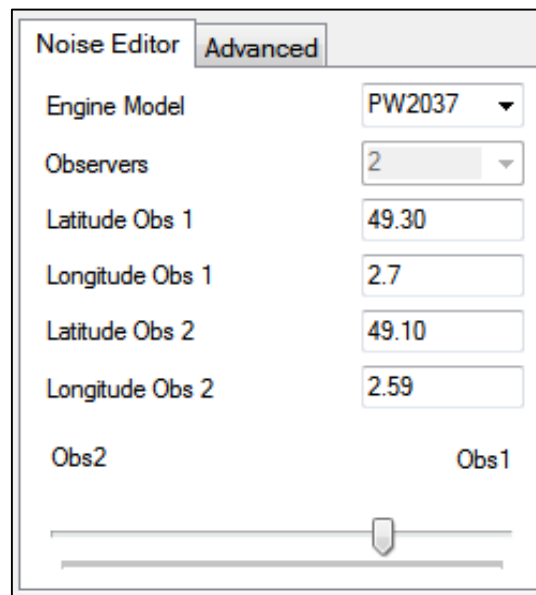
### 4.8.4 Noise Inspector

In previous section 4.7, it has been discussed the process to reduce noise and the noise grid computation. As described in that section, noise computations are performed based on noise data provided by the user. This data is converted into

text files and finally input into 4DT RS Core. The noise inspector is used to input observer's position, noise data file path and priority via *TextBox* controls.

In addition, it is possible to select a noise settings file where empirical data about aircraft engine is provided. This is achieved by making use of a *ComboBox* control.

As detailed in section 4.3.2, Sound Exposure Level (SEL) is measured respect to a reference location named *observer*.



The screenshot shows a software window titled "Noise Editor" with an "Advanced" tab selected. The window contains several input controls:

- Engine Model:** A dropdown menu showing "PW2037".
- Observers:** A dropdown menu showing "2".
- Latitude Obs 1:** A text box containing "49.30".
- Longitude Obs 1:** A text box containing "2.7".
- Latitude Obs 2:** A text box containing "49.10".
- Longitude Obs 2:** A text box containing "2.59".

Below these fields, there are labels "Obs2" and "Obs1" positioned above a horizontal slider bar. The slider bar has a central knob and is currently set to the middle position, indicating equal importance for both observers.

**Figure 4-19: Noise Inspector**

By using the scaling factor feature, it is possible to select which *observer* should receive greater importance level. This *TrackBar* control basically changes the values of  $K_1$  and  $K_2$  in equation (4-19). If scaling factor control is set at the middle, both *observers* are considered to have same importance.

In section 6.3, it is demonstrated a full noise optimization case carried out around London Gatwick Airport (EGKK) that makes use of the *observer's* concept.

Noise model empirical data is specified into a text file that follows a similar format than used by Integrated Noise Model (INM) [66]. This file path read by 4DT RS Core and it is used by the *compute\_noise()* function to calculate SEL using the piecewise function described in previous sections.



Figure 4-20 shows a typical noise settings file used by 4DT RS.

NPD_ID	MET	OP	THR	200	400	630	1000	2000	4000	6300	10000	16000	25000	C
FW4056	E	D	25000	109.8	105.1	101.5	97.3	90.3	82.0	76.0	70.0	62.7	53.9	J
FW4056	E	D	40000	113.0	111.3	105.2	101.5	95.6	88.2	83.1	77.5	70.8	63.3	J
FW4056	M	D	25000	105.3	99.5	93.4	88.0	79.5	70.5	64.3	57.4	49.7	41.5	J
FW4056	M	D	40000	109.1	108.7	97.6	92.5	84.3	75.4	69.3	62.6	55.1	47.2	J
FW4056	S	D	25000	103.7	100.9	96.1	92.7	87.1	80.6	75.8	70.5	64.3	57.5	J
FW4056	S	D	40000	106.8	103.0	100.1	97.1	92.0	85.8	81.0	75.9	69.9	63.4	J
FW2037	E	D	13000	102.9	96.2	95.6	91.6	84.8	76.4	70.1	63.7	56.6	49.5	J
FW2037	E	D	24000	109.4	108.2	101.8	97.8	90.8	82.1	76.2	70.6	64.7	59.4	J
FW2037	E	D	30000	110.8	109.1	103.0	99.6	93.2	84.8	79.2	73.5	67.7	62.2	J
FW2037	E	D	36000	111.5	108.5	103.6	101.2	95.9	88.0	83.0	76.7	70.4	64.0	J
FW2037	S	D	13000	97.4	97.7	90.2	86.8	80.9	73.9	68.6	62.5	55.8	48.8	J
FW2037	S	D	24000	101.5	96.9	94.0	90.4	84.7	77.9	73.2	68.2	62.7	57.0	J
FW2037	S	D	30000	103.3	98.0	96.7	93.6	88.3	81.9	77.1	71.9	66.0	59.9	J
FW2037	S	D	36000	105.5	102.5	100.2	97.7	93.3	87.3	82.3	76.5	69.7	62.6	J

**Figure 4-20:** Noise Specification File

Noise data grid calculated by 4DT RS Core is used to generate noise levels contours. In order to calculate these contours, initially a set of boundaries is defined in the *Noise Level Contour form*. By default, these levels values are defined in Table 4-18.

**Table 4-18:** Noise Level Contours example values

Layer	Sound Exposure Level (dB)
1	30 < SEL < 45
2	55 < SEL < 65
3	70 < SEL < 75
4	85 < SEL

Noise level contour is obtained by making use of a *convex-hull* algorithm described in section 3.6.1, which creates convex polygons by connecting external points. In this way, noise of internal points that are surrounded by the polygon, will be always less or equal to layer's boundary values.

Convex Hull algorithm designed is a modified version of a standard ConvexHull .NET algorithm that has been modified to allow its use with *PointLatLng*-based functions. Basically this algorithm makes use of the following functions:

**Table 4-19:** Convex Hull main functions

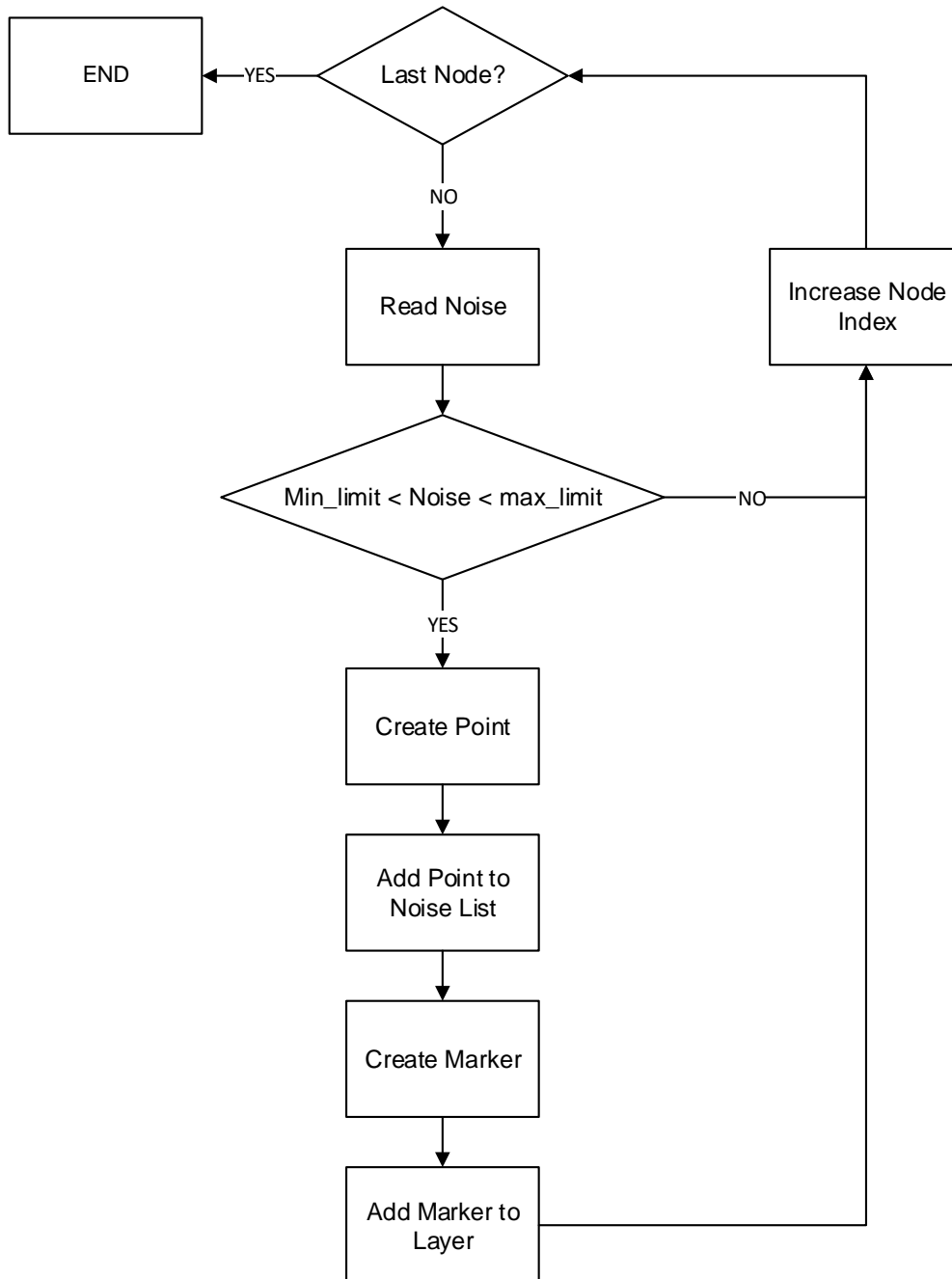
Function	Description
<i>FindConvexPolygon</i>	Creates upper and lower hulls based on a list of <i>PointLatLng</i> objects
<i>ConvexHullCore</i>	Check each <i>PointLatLng</i> object and select them by checking if angle between three (3) points is convex.
<i>IsAngleConvex</i>	Returns true if angle between three(3) <i>PointLatLng</i> points is convex

The main idea of using this algorithm is that it is capable to easily create polygons based on a set of constraints. This function provides the user with capabilities to evaluate the different scenarios by utilizing different noise distributions. Once the upper and lower limits of the layers are selected, a set of function steps are followed to filtering the noise grid data.

Figure 4-21 shows the function *FilterNoiseGrid* which check each noise grid node. If the noise associated to current node is between maximum and minimum noise level selected by the user, the node is saved into a list that contains all the filtered points.

Once noise data is filtered, *PointLatLng* objects are saved into noise list. Finally, a function called *FindConvexPolygon()* is called to obtain the contours for grid points filtered. This process is repeated for each layer.

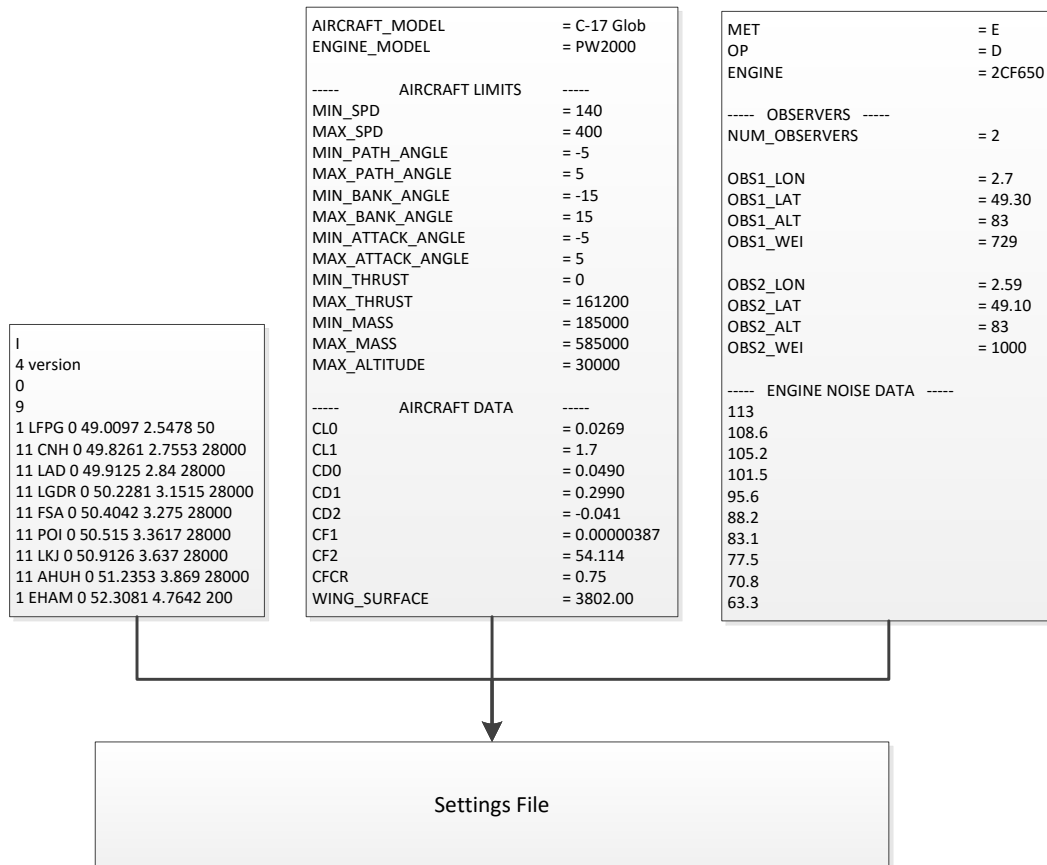
In section 6.5, it is suggested to make use of algorithms that produce concave polygons to obtain more accurate noise contour results.



**Figure 4-21:** FilterNoiseGrid function algorithm

## 4.9 4DT RS Core and 4DT GUI Connection

4DT RS Core is a stand-alone application. Once a testing case has been properly setup, optimization parameters are sent to 4DT RS Core. This process is performed automatically via text files.



**Figure 4-22:** 4DT RS Core and 4DT GUI Connection

As shown in Figure 4-22, flight plan and aircraft files are linked to a unique file named *Settings.dat* file. This means that their paths are written in a general settings file which acts like a connector between 4DT RS Core and GUI. Table 4-20 shows all parameters contained in settings file.

**Table 4-20:** Settings file parameters

Parameter	Possible Value
METHOD	Legendre   Chebyshev
NLP	IPOPT   SNOPT
TYPE	TIME   FUEL   NOISE
FLIGHTPLAN	<i>flightplan.fms</i>
AIRCRAFT	<i>Selected on Aircraft Editor</i>
MAX_NODES	20   40   60   80
MIN_NODES	10

MAX_ITER	2000+
TOLERANCE	0.01<
INITIAL_MASS	$max\_mass > value > min\_mass$
INITIAL_HEADING	$355 > value > 0$
INITIAL_VELOCITY	$max\_speed > value > min\_speed$

These settings are loaded into 4DT RS Core to setups the optimal control problem. Noise data is processed only if the value of *TYPE* variable is equal to string “*NOISE*”.

Once optimization process is completed, 4DT RS Core exports the states, control variables, noise grid and noise perceived at each observer into result files. Finally 4DT GUI uses these files to plot results and calculating flight information.

## 4.10 Computing Flight Information

Despite of 4DT RS main purpose is generating optimal trajectories; some functions have been developed in order to compute additional and complementary information about the flight. These features were designed in order to provide the user with top-level or management information related to consumed fuel or total flight costs that could be used to compare with baseline. Figure 4-23 shows a typical representation of flight information calculated by 4DT RS.

Results			
Name	Latitude (deg)	Longitude (deg)	Time (min)
▶ TOC	49.76	2.72	8.36
BOD	51.25	3.88	25.68
Total Time (min)	38.06	Distance (nm)	273.48
Fuel Cons (lb/h)	12811.83	Fuel Cons (gal)	1213.13
Fuel Cons (gal/m)	4.44	Fuel Cons (lb)	8128
Price per hour Flight	6692.74	Fuel Cost (usd)	4245.97

**Figure 4-23: Flight Information**

#### 4.10.1 Fuel, flight cost, distance and maximum SEL

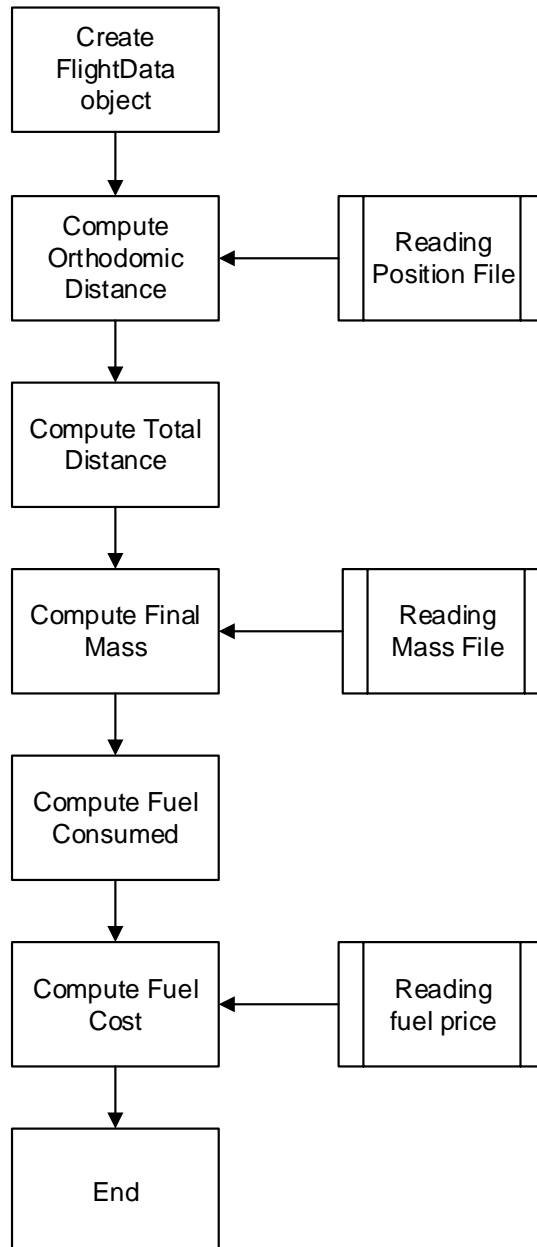
In order to calculate fuel, flight cost and distance, it has been implemented an object type *FlightData()* which is composed by the elements shown in Table 4-21:

**Table 4-21:** *FlightData* struct

Variable	Type	Description
Fuel Cost	double	$Cost_{usd} = Fuel_{gal} \times Cost_{per\ gal}$
Fuel Consumed (gallons)	double	$Fuel_{gal} = \frac{M_i - M_f}{6,79}$
Fuel Consumption Rate (lb / hr)	double	$Fuel_{lb/hr} = \frac{(M_i - M_f) \times 60}{T_{total}}$
Fuel Consumption Rate (gal / n. miles)	double	$Fuel_{gal/mile} = \frac{Fuel_{gal}}{D_{total}}$
Fuel Cost Rate (US dollars / hr)	double	$Cost_{usd/hr} = \frac{Cost_{usd} \times 60}{T_{total}}$
Orthodomic Distance (n. miles)	double	$Distance_{mile} = \sqrt{D_{east}^2 + D_{north}^2}$

Additional flight data calculation is carried out by *ComputeFlightData()* function, which basically receives time, mass, latitude and longitude vectors from 4DT RS Core solution and uses equations described in Table 4-21 to calculate the relevant flight data. Values are exported in an object type *FlightData()*.

Figure 4-24 shows the *ComputeFlightData* function algorithm. The function makes use of text files mass, latitude and longitude as well as fuel price (input by user in the aircraft Inspector).



**Figure 4-24:** ComputeFlightData function algorithm

#### 4.10.2 Predicted Time of Arrival

One of the most important characteristics of 4D trajectories is the presence of a Required Time of Arrival (RTA). This parameter would be commonly assigned by Air Traffic Controller (ATC). Once this parameter is set, it is expected to take place a *negotiation process* between crew and ATC.

From one side, ATC must manage airspace based on this required time of arrival. On the other side, crew must negotiate the controlled time of arrival so it could be reached in order to avoid holding patterns while optimizing the trajectory.

Predicted Time of Arrival is calculated by 4DT RS in order to give to crew a *clue* about the capacities of aircraft performance given that aircraft should be capable to reach each waypoint at proposed predicted time of arrival.

*ComputeTimeOfArrival()* function makes use of time, latitude and longitude vectors obtained from optimal trajectory and flight plan input by user.

This function calculates predicted time of arrival by obtaining the flight time at the point where the following pair of inequalities is *true*:

$$\Delta\phi_i \leq \varepsilon$$

$$\Delta\lambda_i \leq \varepsilon$$

where  $\varepsilon$  is the radius of a circle centred at waypoint  $i$  (minimum fly-by distance) and  $\Delta\phi$  and  $\Delta\lambda$  refers to error distance between aircraft and flight plan waypoint  $i$ .

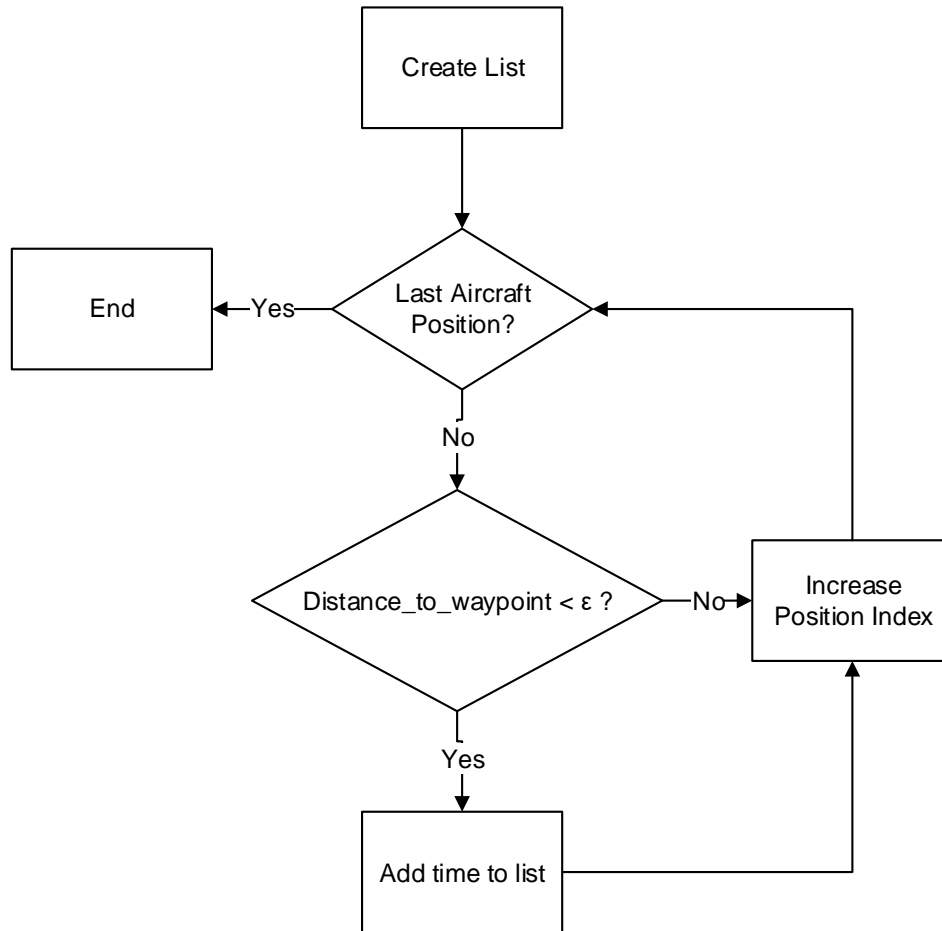
$$\Delta\phi_i = |\phi_i - \phi_{fp}| \quad (4-24)$$

$$\Delta\lambda_i = |\lambda_i - \lambda_{fp}| \quad (4-25)$$

The time of arrival is exported as a *List<double>* object and is loaded into a *DataGridView* control.

Figure 4-25 shows the *ComputeTimeOfArrival* function algorithm which return a list with all predicted time of arrival values.





**Figure 4-25: Compute Time of Arrival Algorithm**

Figure 4-26 shows a typical result of predicted time of arrival computed by the software after an optimal solution has been found.

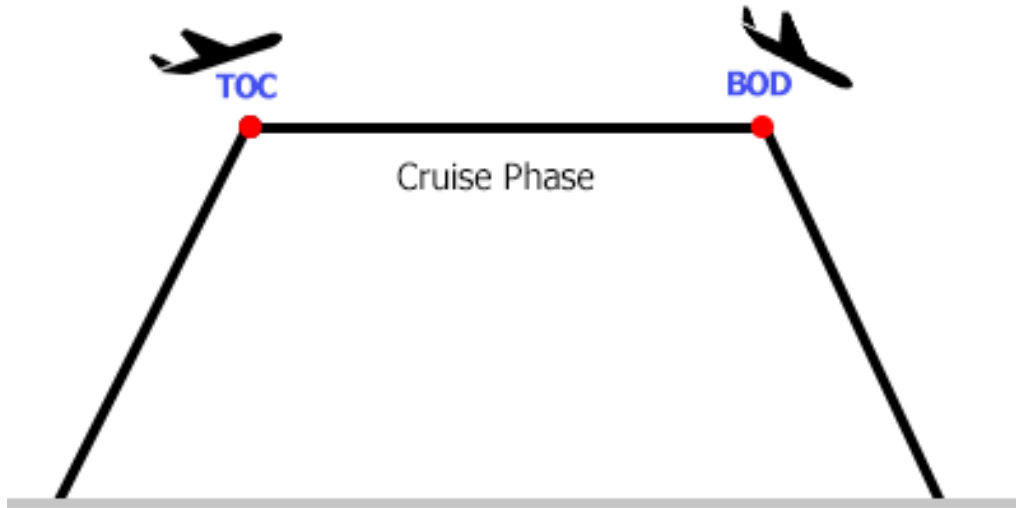
#	Name	Latitude (deg)	Longitude (deg)	Altitude (ft)	Time (min)
1	LFPG	49.0097	2.5478	50	0
2	CNH	49.8261	2.7553	28000	8.15
3	LAD	49.9125	2.84	28000	8.94
4	LGDR	50.2281	3.1515	28000	13.07
5	FSA	50.4042	3.275	28000	15.09

**Figure 4-26: Predicted Time of Arrival**

#### 4.10.3 Top of Climb and Begin of Decent

Top of climb (TOC) point can be defined as the location where aircraft altitude is equal to cruise altitude. At this point, cruise phase begins, hence thrust levels and fuel flow decrease and aircraft is maintained at *steady-state*. This is considered an important point for flight planning and navigation; also for ATC, this means

that aircraft altitude and speed becomes constant, which could affect the way aircraft is processed in airspace management (air traffic controllers decrease their stress when aircraft altitude and speed are constants). Figure 4-27 shows top of climb point.



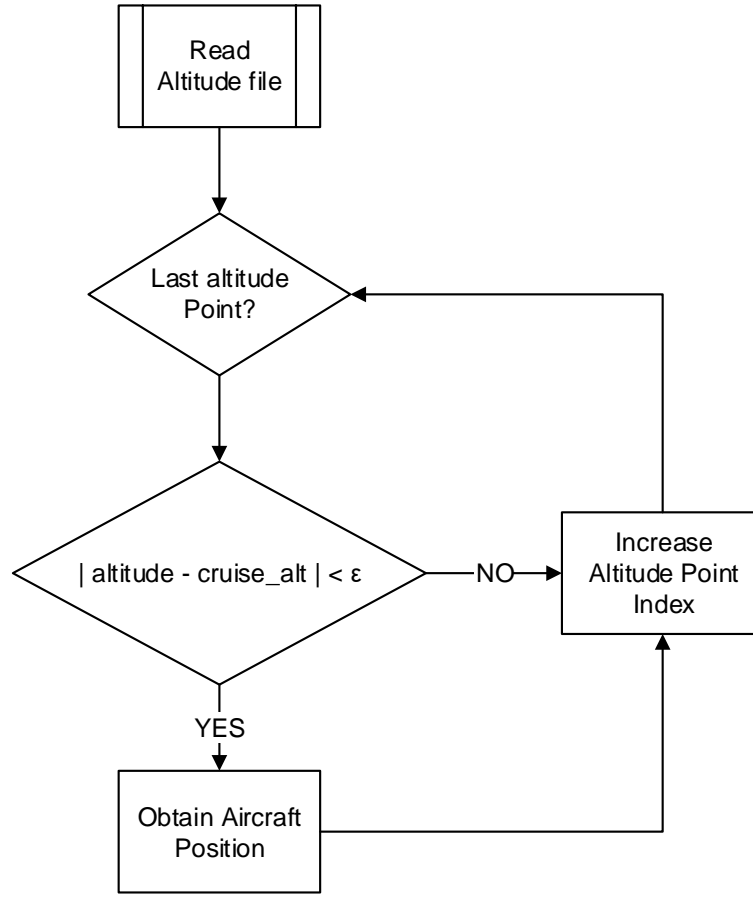
**Figure 4-27:** Top of Climb (TOC) and Begin of Descent (BOD)

4DT RS calculates latitude, longitude and time when:

$$|z - z_{cruise}| < \varepsilon_z$$

where  $z$  and  $\varepsilon_z$  are aircraft altitude and altitude tolerance respectively. By default the value of tolerance  $\varepsilon_z$  has been set to 50 ft.

This logic has been implemented in the function *ComputeBODTOCPoints()* that basically goes through position vectors and applies the expression described above. A section of this function is shown in the following figure.



**Figure 4-28:** *ComputeBODTOCPoints* function algorithm (Part 1)

Begin of descent (BOD) point is defined as the location where cruise phase is completed. Therefore aircraft altitude decreases to approach phase altitude or final approach altitude (in case of a Continuous Descent Approach (CDA) procedure is performed). Thrust usually is set to idle and speed decreases. Figure 4-27 shows begin of descent point.

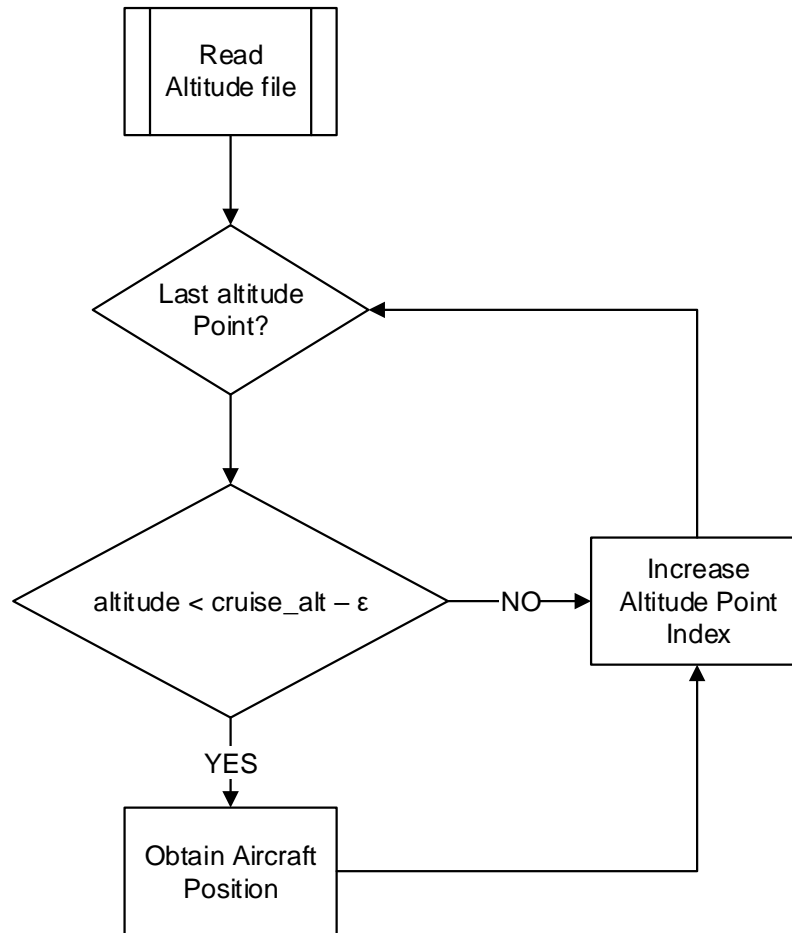
This point is calculated by using the following equation:

$$|z_i - z_{cruise}| \geq \varepsilon_z \text{ and } t_i > t_{TOC}$$

where  $i$  represents the index of the current waypoint and  $z_{cruise}$  is the cruise altitude defined in the flight plan.

By following the same procedure than TOC point, the second part of function *ComputeBODTOCPoints()* goes through position vectors to compute BOD point.

The index of variable BOD starts in the value of the TOC index variable. This is done in order to ensure that  $t_i > t_{TOC}$ .



**Figure 4-29:** *ComputeBODTOCPoints* function algorithm (Part 2)

## 4.11 Trajectory Representation

Optimal values for states and control variables, noise and time data is obtained from data files exported by 4DT RS Core. These files are used to create several plots.

First, it is used the function *PlotTrajectory()* described in section 4.8.1 to plot the horizontal trajectory (longitude vs latitude) in the *GMapControl*.

Additionally, it is used the libraries *ZedGraph* [73] to produce a set of scientific plots that provides extra features to the user such as zoom, detailed values, set axis scales or export data to image files.

The base control of these libraries is *ZedGraphControl*, which is an element that is used to link a set of lists of type *List<double>* to a graphic plot included in the 4DT RS GUI.

Since a *ZedGraphControl* object can be associated to one or more plots, this information is given to control by another object named *GraphPane*. The *GraphPane* object contains properties relative to plot window such as title, axis labels or plot size. The data to be plotted (the trajectory curve) is represented by another object of type *LineItem*. This object contains properties of curve such as smoothness, colour, line width and tension.

In summary, it has been used the following *ZedGraph* objects.

**Table 4-22:** ZedGraph objects

Element	Object	Description
Control	<i>ZedGraphControl</i>	It is the control where data is plot.
Plot	<i>GraphPane</i>	It is an object that contains plot information
Curve	<i>LineItem</i>	Curve or line to be plotted

Different plots are created for each trajectory. Horizontal and vertical trajectories show aircraft situation in different planes. States and control variables shows aircraft status with respect to time. For this reason, it has been implemented the function *PlotGraph()* that is generic for any type of plot on a *ZedGraphControl* element. This function receives all the parameters needed to create a plot such as title, axis labels and type of plot symbol.

However, the lists of type *List<double>* are previously created by pre-processing functions that simply read the relevant files exported by 4DT RS Core (e.g. heading.dat, altitude.dat or time.dat) and convert data into a list.

This pre-processing functions also convert angles from radians to degrees and validate the data contained in trajectory files.

**Function:** PlotGraph

**Inputs:** x\_list, y\_list, title, x\_label, y\_label, symbol, zed\_control

**Output:** n/a

- 1) **Create Pane**  
Control.GraphPane;
- 2) **Settings titles and labels**  
Pane.Title.Text = title;  
Pane.XAxis.Title.Text = xtitle;  
Pane.YAxis.Title.Text = ytitle;
- 3) **Add the curve**  
LineItem curve;  
curve = Pane.AddCurve(title, x\_arr, y\_arr, color, symbol);  
curve.Line.Width = 1.5F;
- 4) **Make the curve smooth with cardinal splines**  
curve.Line.IsSmooth = true;  
curve.Line.SmoothTension = 0.6F;
- 5) **Fill the symbols with white to make them opaque**  
curve.Symbol.Fill = new Fill(Color.White);  
curve.Symbol.Size = 10;
- 6) **End**

**Figure 4-30:** *PlotGraph* function example in pseudo code and C#

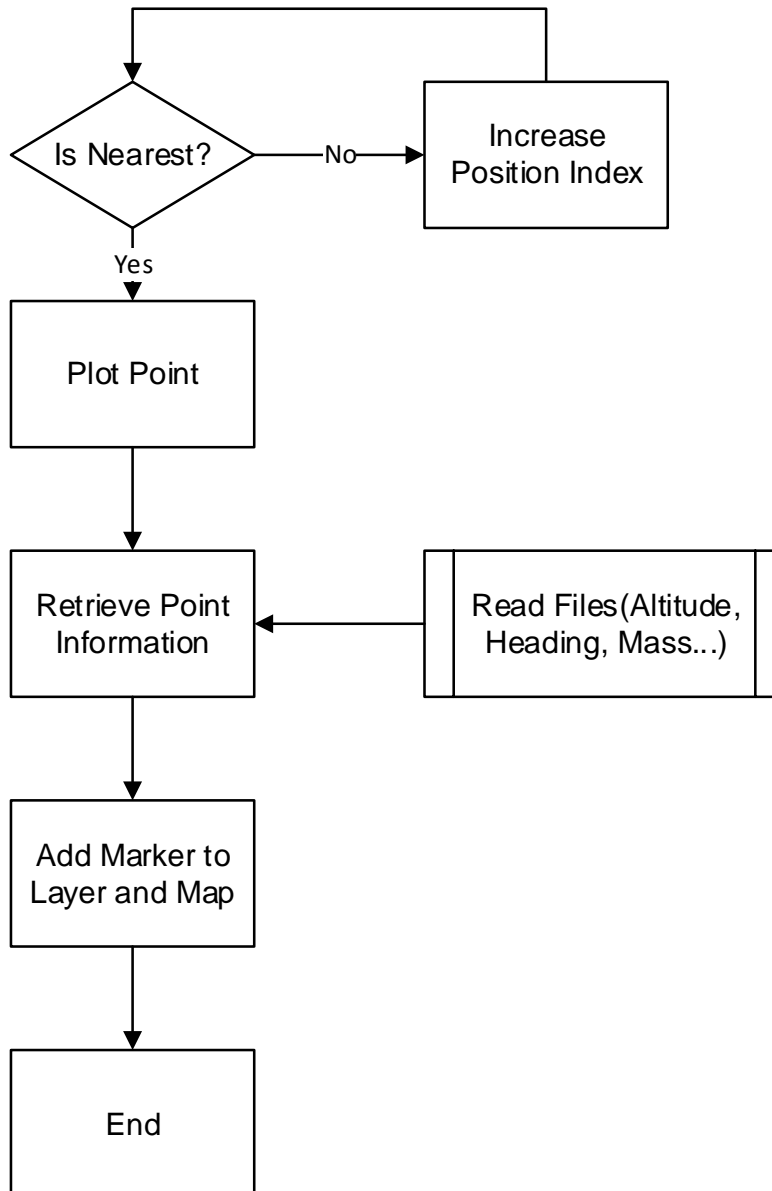
#### 4.11.1 Detailed View Tool

4DT RS has been provided with a special tool that allows checking aircraft status parameters such as fuel consumed, time, heading and altitude with respect to its position.

Behind *Detailed View* tool there is a set of functions that plot points for each optimization node and retrieves automatically data from different sources relative to this point.

This data is shown when user moves mouse cursor from one node to another, however, since there is limited number of points, it is necessary to find the nearest point to cursor position.

Functions *FindNearestPoint()* and *FindIndexPositionVector()* are used to obtain this position and retrieve interesting data from trajectory result files.



**Figure 4-31:** FindNearestPoint function algorithm

An object *PointLatLng* is obtained from the function *GMapMouseMove()* which return the mouse position on the map. Then, it is used the function *IsNear()* to obtain nearest trajectory point to mouse position. Finally, trajectory information such as altitude, fuel consumed or heading is obtained from returned point. Information is shown using a *tooltip* property contained in *GMapMarker* objects.

Figure 4-32 shows the “Detailed View” tool. Note that blue dots represent the optimal collocation points, and the box is showing relevant data relative to the green (selected) point.



**Figure 4-32:** Detailed View Tool

From the user point-of-view this is a really powerful tool to understand aspects of the optimal trajectory and the discretization process, for example, the *tooltip* shows where collocation nodes have been located during optimization process. But also shows the fuel burn by aircraft from take-off to selected point or aircraft heading at that moment.

This information could be interpolated to guess aircraft status at each point of the optimal trajectory or alternatively, the number of nodes could be increased to obtain more accurate information.

## 4.12 Preferences and Help

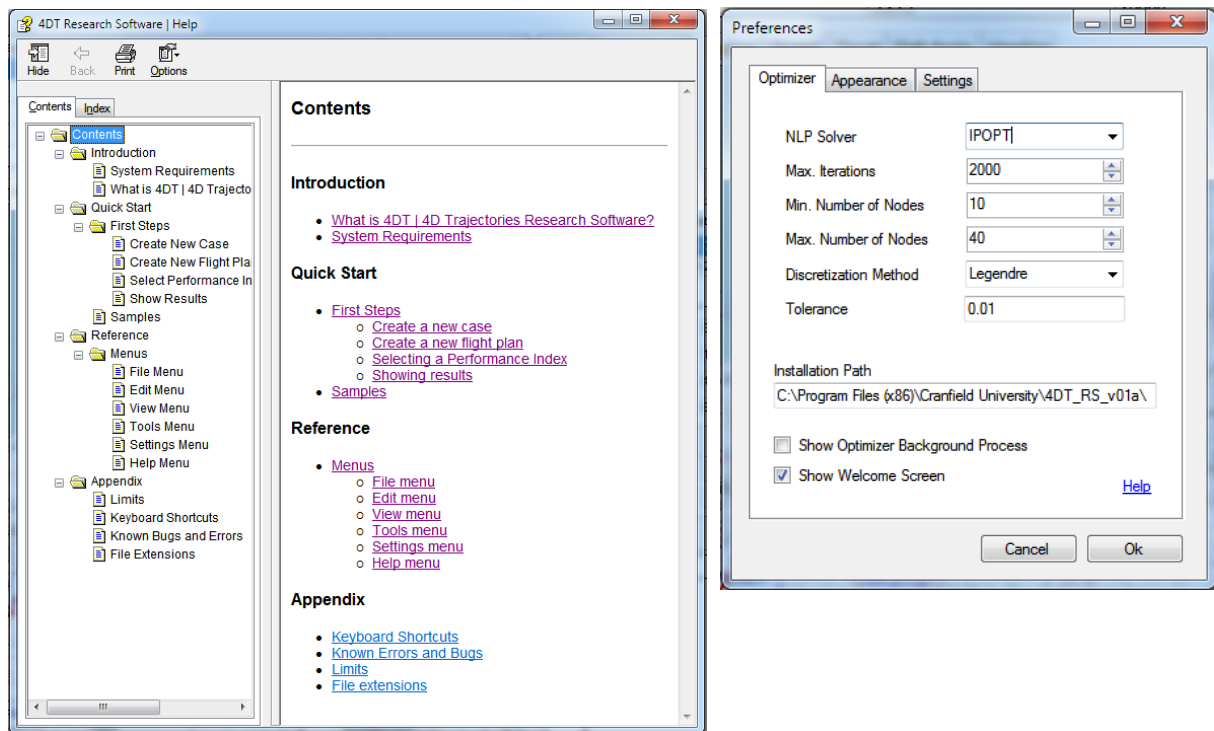
Preferences provides control of configuration parameters to the user. Preferences form receives PSOPT configuration preferences on *TextBox* controls. The data is saved/loaded to/from a text file named *preferences file*. From this form it is possible to change almost all parameters mentioned in previous section 4.7.



Also it is possible to switch on/off the capability to show the optimization calculation process in a *command symbol system* window instead of running 4DT RS Core in background.

In addition, a complete Help section with pre-defined samples cases has been developed and implemented in HTML code, so user can easily get access to required information. It has been created a set of HTML files for each section contained on the help. Then it has been compiled into a *Microsoft Compiled HTML* help file that is loaded when user click on help button or alternatively press F1 key shortcut.

Preferences and Help forms is shown in the following figure.



**Figure 4-33: Preferences and Help**

# ***Chapter 5***

## **TRAJECTORY TRACKING & GUIDANCE**

### **5.1 Introduction**

This chapter describes the trajectory Tracking and Guidance system that has been developed as a complementary module for 4DT RS. It covers a system overview in section 5.2, tracking system design is described in section 5.3, guidance system in section 5.4, avionics systems indicators are covered in section 5.5, a simulation framework used to test initial versions of the tracking and guidance system is described in section 5.6. Finally the T&G module for 4DT RS is described in section 5.7

### **5.2 System Overview**

Tracking & Guidance (T&G) system module aims at testing and validating the trajectories generated by 4D Trajectories Research Software v0.1a (4DT RS).

Two main objectives are pursuit in this tracking and guidance system:

1. Provide pilots with guidance by making use of visual indicators located in Primary Flight Display (PFD) and Navigation Display (ND).
2. Provide aircraft with automatic guidance based on heading, altitude and vertical speed. These values are aimed to be used in the aircraft automatic flight control system.

The system initially calculates the cross-track error and altitude error based on the aircraft position and the reference trajectory (generated by 4DT RS). If cross-

track error is greater than a specified tolerance, it computes vertical and horizontal guidance commands to allow aircraft to follow the reference trajectory.

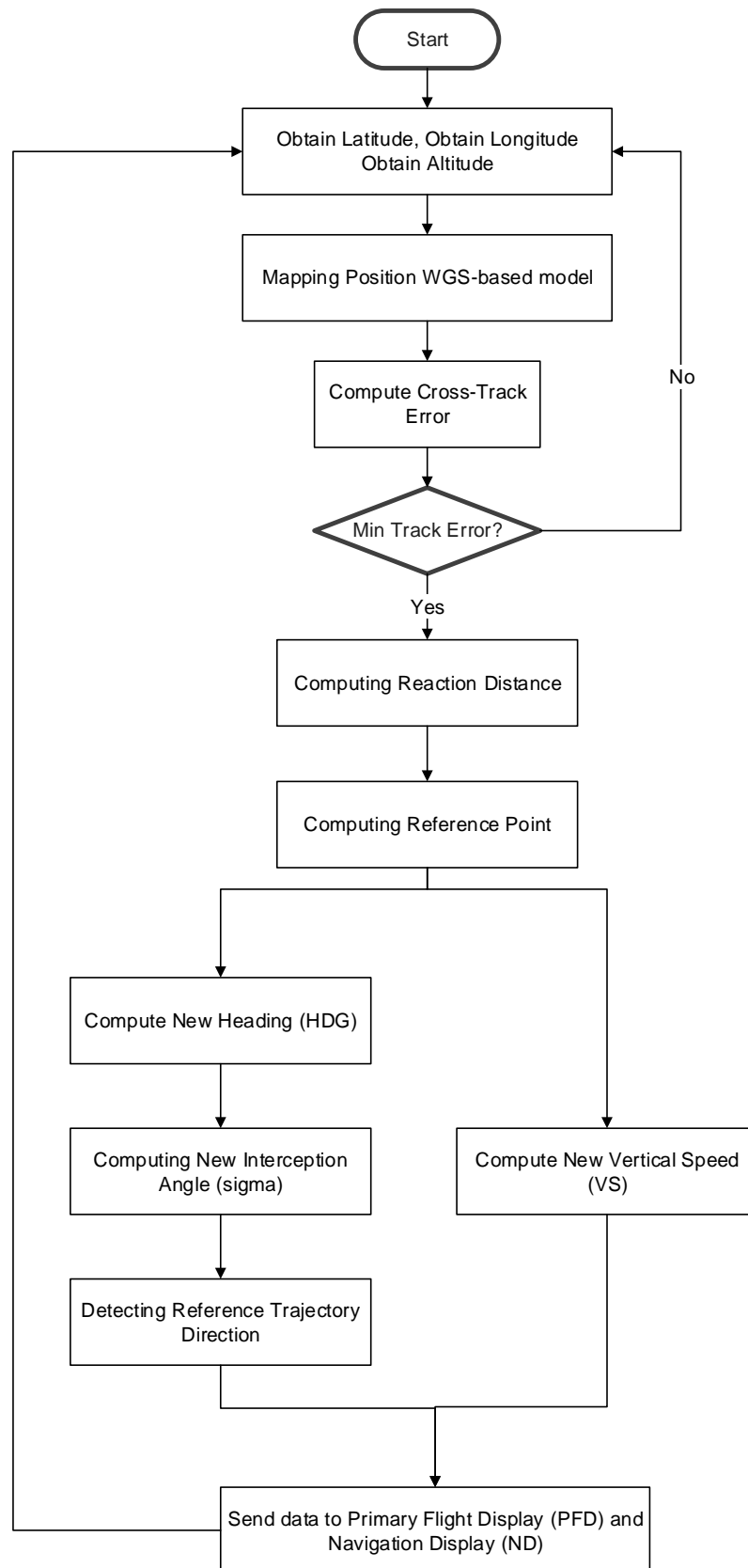
Guidance commands are sent to autopilot system, therefore horizontal guidance command is aircraft's heading which is also displayed in the Navigation Display (ND) and vertical guidance command is vertical speed and altitude which are displayed in the Primary Flight Display (PFD).

Figure 2-6 shows the Tracking and Guidance module's main control loop (Also a complete schema of this system can be found in Appendix B).

In order to properly describe this system, it has been divided into two sections: Tracking system which describes how errors are computed and Guidance system which describes how vertical and horizontal control commands are calculated and sent to autopilot system.

Early versions of tracking & guidance system, has been implemented using MATLAB®. For this, it has been developed a simulation framework that describes aircraft dynamics for lateral guidance. The framework used simple equations for point displacement in flat surface.

Furthermore, in order to obtain more accurate results, the simple equations have been replaced by realistic aircraft dynamics by creating a C# guidance module and linking the guidance system to a commercial flight simulator via User Datagram Protocol (UDP). The result is a stand-alone guidance module that can be executed once a reference trajectory is generated by 4DT RS. Therefore, the reference trajectory can be simulated and evaluated in a more realistic environment.



**Figure 5-1: Tracking & Guidance System Main Control Loop**

### 5.3 Tracking System

Tracking system has been designed to maintain a Required Navigation Performance (RNP) condition. According to main control loop shown in Figure 5-1, first algorithm step is receiving aircraft position. It is expected that the tracking system make use of aircraft GNSS capabilities in order to obtain aircraft latitude, longitude and altitude. This step relies on an *ideal* GNSS system, this means that errors introduced by GNSS related to atmosphere conditions, distance or surface for computing aircraft position has been neglected during the designing process.

Following same strategy used on 4DT RS trajectories, geographic coordinates system are converted to North-East coordinate system by using WGS-based function described in section 4.7.

Therefore, it is possible to compute cross-track error in distance units by using orthodomic distance equations. Cross-track error is defined by the minimum value of  $e_i$ :

$$e_i = \sqrt{(x_i - x_{air})^2 + (y_i - y_{air})^2} \quad (5-1)$$

given that  $i \in [0, N]$  and  $N = N_{nodes} \times WP$

where.

$x_i$  and  $y_i$  are the east and north distances of a reference node  $i$

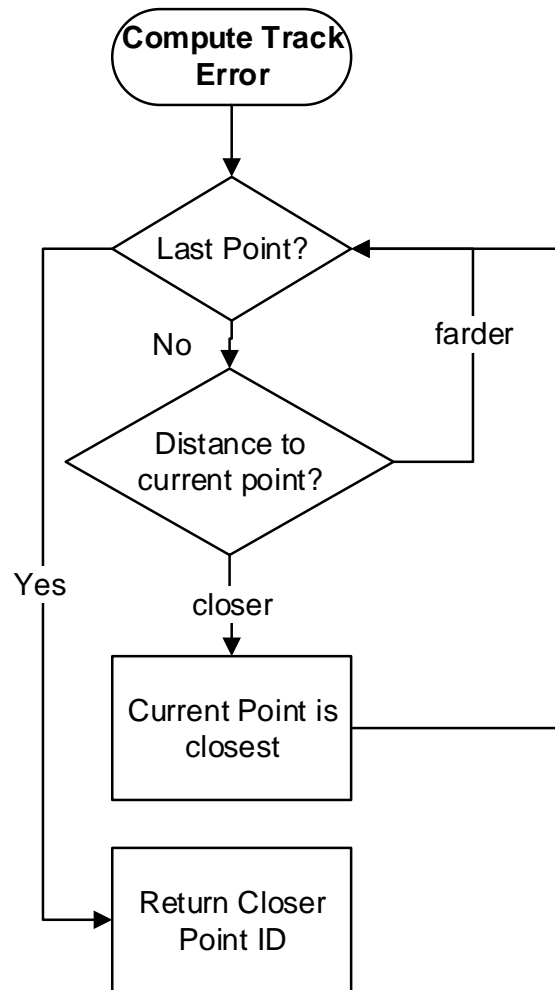
$x_{air}$  and  $y_{air}$  are aircraft east and north distances

$N_{nodes}$  is the number of collocation points used to solve the optimal control problem.

$WP$  is the number of waypoints of flight plan

Note that  $N_{nodes}$  value correspond to size of vectors exported by 4DT RS that defines aircraft lateral position.

Computing cross-track error sub-loop tries to find the closest point to aircraft position by making use of equation (5-1). This algorithm is shown in Figure 5-2.



**Figure 5-2:** Compute track error loop

Once the closest point is obtained (cross-track error node), it is computed the altitude error. To compute this error it is used the predicted trajectory exported by 4DT RS. Altitude error is computed by comparing aircraft altitude with cross-track error node's altitude. In this way, altitude error is calculated using the following expression:

$$e_{alt} = z_i - z_{air} \quad (5-2)$$

where,

$z_i$  is the cross-track error node altitude

$z_{air}$  is the aircraft altitude

Note that if aircraft altitude is greater than reference path altitude, altitude error is negative. This peculiarity is used when computing vertical speed guidance command.

## 5.4 Guidance System

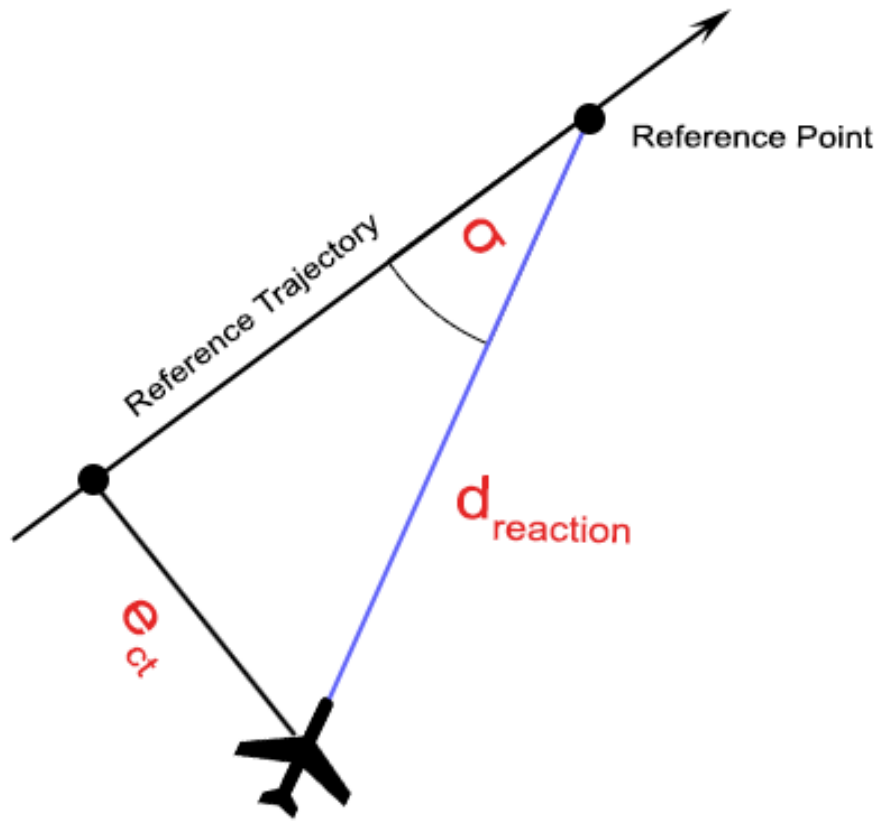
The guidance system accepts the tracking errors and generates guidance commands according the types of tracking errors. It has been designed to only get triggered if minimum track-error or altitude error conditions are exceeded beyond a defined tolerance.

If the tracking errors are greater than defined tolerance, the guidance commands for lateral and vertical trajectories are calculated based on two different loops. Figure 5-1 shows that lateral and vertical loops.

### 5.4.1 Lateral Guidance

Lateral guidance has been designed inspired on the *Proportional Navigation* concept discussed in section 3.5. A simple *static target point* approach has been used. Aircraft acceleration is not taken into account thus is considered aircraft speed is constant.

The main idea relies on computing the aircraft heading to follow a *target (reference point)* that is selected based on the distance  $d_{reaction}$  which is the distance between aircraft and the interception point.



**Figure 5-3:** Lateral guidance based on static reference point

The distance  $d_{reaction}$  is variable and depends on the interception angle ( $\sigma$ ) which also varies proportionally to the cross-track error ( $e_{ct}$ ).

$$d_{reaction} = \frac{e_{ct}}{\sin(\sigma)} \quad (5-3)$$

Figure 5-3 shows a typical situation where aircraft is off of reference trajectory. This figure shows cross-track error computed by tracking system. Note that reference point position must change when reaction distance or interception angle change(s).

Guidance has been designed in order to intercept reference as soon as possible. For this reason, interception angle changes proportionally to cross-track error. It is calculated using the following expression:



$$\sigma = \min((Kp \cdot e_{ct}), \sigma_{max}) \quad (5-4)$$

where,

$Kp$  = Proportional Control Factor

$e_{ct}$  = Cross-track error

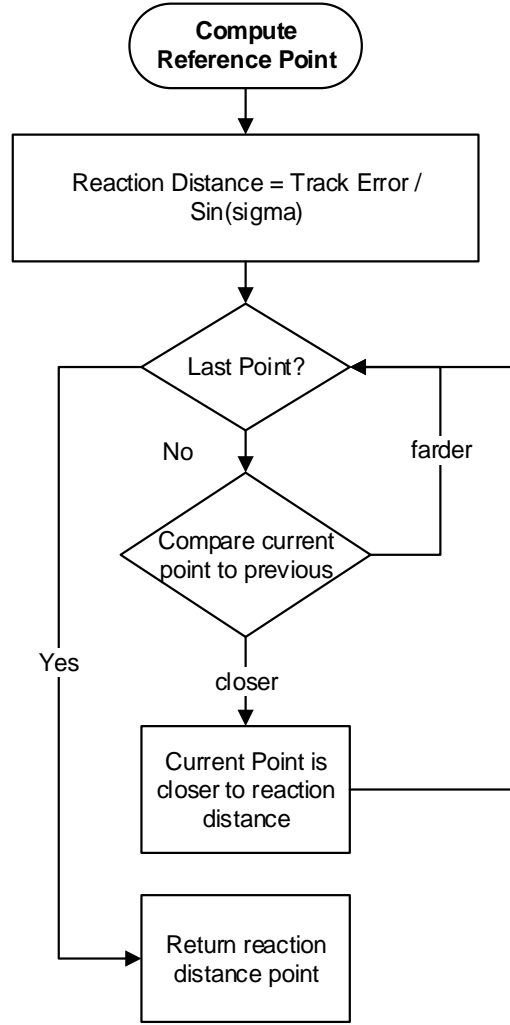
$\sigma_{min}$  = Minimum interception angle

This means that if aircraft position is considered *far* from reference trajectory, interception angle will be more perpendicular to reference trajectory and therefore reference node will be closer to cross-track error node. In the opposite situation, in case aircraft is at a position considered *close* from reference trajectory, this interception angle becomes narrower.

For this particular case, the concept of “far” and “close” is directly defined by the constant  $Kp$ , which is empirically adjusted depending of the guidance system desired behaviour.

A design value  $\sigma_{max}$  is defined to limit the maximum interception angle available (e.g. interception angle could be limited to 30 degrees).

Once reaction distance is obtained, it is possible to obtain reference point based the algorithm shown in Figure 5-4.



**Figure 5-4:** Compute reference point algorithm

Algorithm represented in previous figure aims at finding the node/point  $(x_{ref}, y_{ref})$  that closest matches the reaction distance  $(d_{reaction})$  inside the position vectors.

Commands for lateral guidance are represented by aircraft heading. This heading is computed using the following expression:

$$\varphi = \left( \frac{180}{\pi} \cdot \tan^{-1} \left( \frac{y_{air} - y_{ref}}{x_{air} - x_{ref}} \right) \right) - 90 \quad (5-5)$$

where,

*air* = current aircraft position

*ref* = reference point position

Once heading command is calculated, a new interception angle is calculated for next loop iteration.

#### 5.4.2 Vertical Guidance

Vertical guidance has been designed in order to generate vertical speed commands so aircraft maintain reference altitude. Unlike lateral guidance, altitude reference point is not located far in advance and does not depends of interception angle.

Once tracking system computes cross-track error, an altitude reference value is obtained from the node next to cross-track error. This is the most immediate (and possibly parallel to aircraft position) node.

The vertical guidance command is defined by aircraft vertical speed required to achieve the reference altitude proportionally to a design factor ( $Ks$ ). Therefore, vertical speed command is calculated based on equation:

$$Vs = Ks \cdot (z_{i+1} - z_{air}) \quad (5-6)$$

where,

$Ks$  = Proportional Vertical Control Factor

$i$  = index of current cross-track error reference error

$air$  = refers to current aircraft vertical position

Following same design line used in previous section, vertical guidance command is limited by design factors depending of aircraft performance. The following interval has been taken into account for vertical guidance.

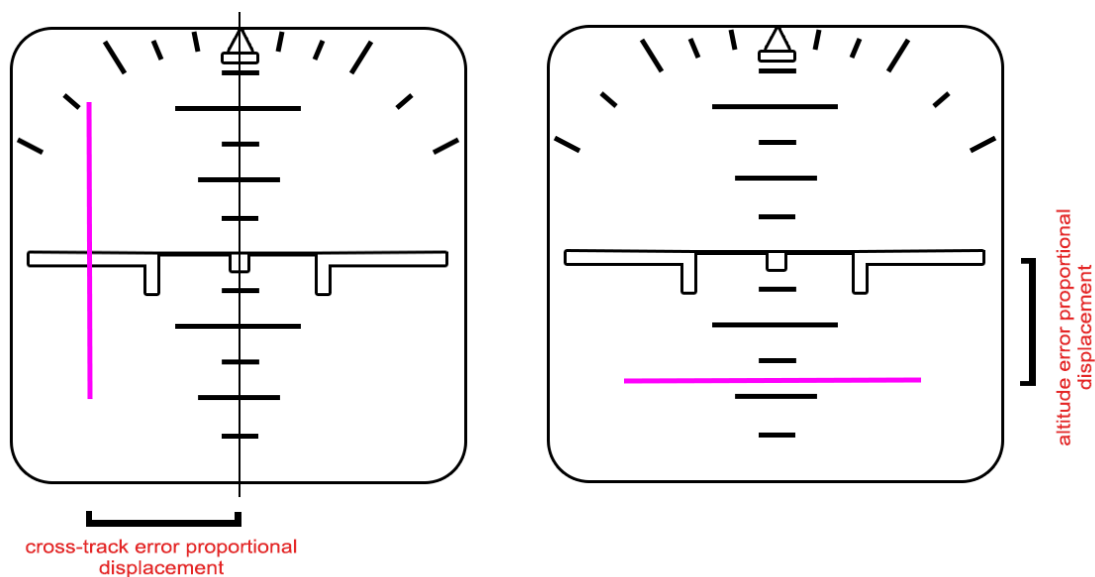
$$Vs \in [Vs_{min}, Vs_{max}]$$

## 5.5 Avionics Systems Indicators

Guidance commands are usually linked to aircraft flight control system or autopilot system. However, visual indicators should be used in order to show the correct path to crew. This section aims at explain a proposed solution for this problem.

Lateral guidance command is represented by aircraft heading. However, this command does not indicate how far the reference point is with respect to aircraft position. Similarly, vertical guidance command is represented by vertical speed, but an explicit difference between aircraft altitude and reference altitude should be indicated to crew.

Following the same strategy used in Instrument Landing Systems (ILS) in modern aircraft [34], reference trajectory is indicated using lateral and vertical bars located on the Attitude Indicator (AI) on the Primary Flight Display (PFD) as shown in Figure 5-5.



**Figure 5-5:** Cross-track and altitude error indicators in PFD

For lateral guidance, a vertical bar moves proportionally to cross-track error from left-to-right or in the other way. In order to obtain this effect, aircraft relative position with respect to current path is calculated in main guidance loop (Figure 5-1).

In order to calculate the relative position, it is used *the sign of  $lr$* , which is the determinant of vectors  $(\overline{AB}, \overline{AM})$  where  $M(x, y)$  is the aircraft position and  $A$  and  $B$  are the points that compose the path. This determinant is calculated using following expression:

$$lr = (x_{ref} - x_{i-1}) \cdot (y_{air} - y_{i-1}) - (y_{ref} - y_{i-1}) \cdot (x_{air} - x_{i-1}) \quad (5-7)$$

where,

$i$  = index of current cross-track error reference error

$x$  = East position

$y$  = North position

$air$  = refers to current aircraft position

$ref$  = reference point position

## 5.6 MATLAB® Simulation Framework

First part of guidance module has been implemented and tested using MATLAB®. For this, a rapid simulation environment for unit testing has been created.

This simulation framework has been created using simple equations for point displacement in flat surface as follows:

$$x = x_{air} + V \sin(\varphi) \quad (5-8)$$

$$y = y_{air} + V \cos(\varphi) \quad (5-9)$$

where,

$x$  = aircraft new north distance

$y$  = aircraft new east distance

$air$  = refers to current aircraft

$V$  = aircraft speed

$\varphi$  = aircraft heading

This framework has been used to test preliminary aspects of tracking system such as cross-track error computation and features of guidance system such as heading, reaction distance and reference point calculation.

Complete code of this framework as well as an initial version of Tracking & Guidance system can be found in Appendix D.

## **5.7 4DT RS Guidance Module**

The 4DT RS Guidance module has been coded in C# and compiled to support Windows® operating system.

This guidance module is composed by the following components:

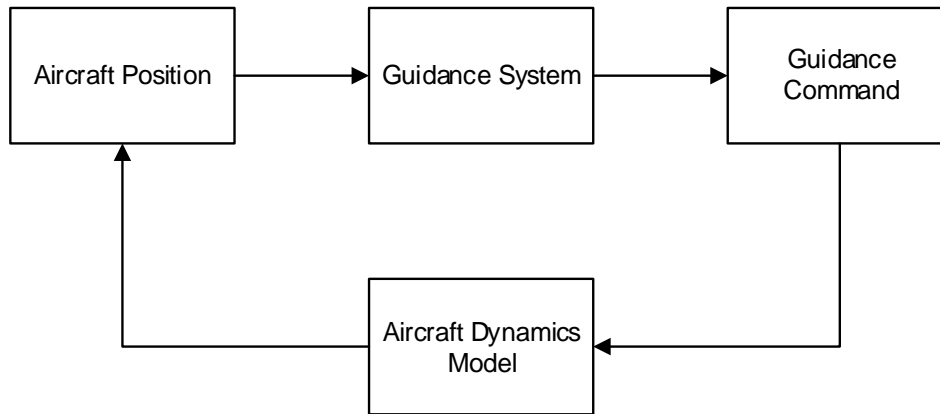
- 1) Guidance System Core
- 2) 4DT RS Guidance Graphic User Interface (GUI)
- 3) X-Plane® UDP Communication interface

### **5.7.1 Guidance System Core**

Guidance system core is based on the implementation of the guidance algorithm explained in previous section.

The core is composed by functions that synthetize aircraft position from GNSS system, a main control loop makes use of a *timer* element [68]. This means that each iteration is executed at each *timer\_tick* event and finally a set of guidance commands which is composed by heading, vertical speed, reference altitude and vertical/horizontal displacement for attitude indicator bars.

Figure 5-6 shows the main control loop and its interaction with aircraft dynamics model.



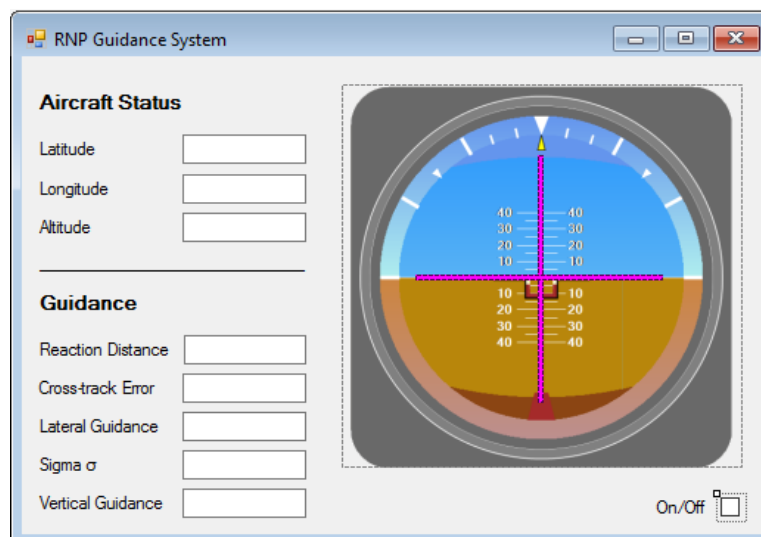
**Figure 5-6:** Top-level view of tracking & guidance system

Note that basically, the guidance system core involves all the components of the guidance system excepting those that are related to aircraft dynamics.

The guidance system has been designed to use any aircraft model. However, for practical and more realistic purposes this model has been replaced with data received from a commercial flight simulator.

### 5.7.2 4DT RS Guidance Graphic User Interface (GUI)

From the user point-of-view, the guidance module is shown as window which contains information relevant to Tracking & Guidance system.



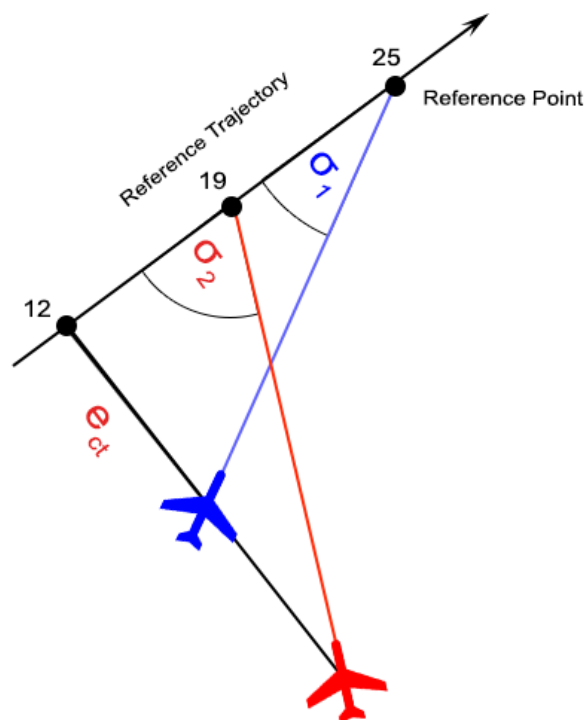
**Figure 5-7:** Tracking & Guidance System GUI

The graphic user interface is divided in three sections: aircraft status, guidance and Attitude Indicator (AI) simulation.

First section shows relevant information regarding to aircraft position, this is latitude, longitude and altitude.

Second section shows information related to tracking and guidance. Cross-track error is expressed in nautical miles. Next to cross-track error and reaction distance fields, the node number in equivalent to its vectors position is shown between brackets.

For example, if cross-track error field shows “5 (12)” this means that aircraft position is 5 nautical miles off reference trajectory. This distance is being measured with respect to node in position 12 of reference trajectory vector. This information is quite useful specially for showing how reference point changes to a farer position when cross-track error is close to RNP condition.



**Figure 5-8:** Example of reference point selection



An example of this effect is shown in following Figure 5-8, where it is visible that *blue* aircraft heads to a reference point located farer from cross-track node in comparison with *red* aircraft's reference point. In practical environment this effect is responsible for creating an arc-shaped path by aircraft when flying back to reference trajectory.

Third section is represented by indicators proposed to be shown in Attitude Indicator (AI) of Primary Flight Display (PFD). Indicators are represented by two bars that move horizontally and vertically in proportion to cross-track error and altitude error respectively.

For this effect, it has been implemented a function that moves *bar images* a defined number of pixels in up/down or right/left directions, depending of the value of errors computed by tracking system. Obviously, this movement has been limited to the attitude indicator area.

Finally, an *on/off* button represented by a *checkbox* object has been added in order to provide more control to the user about guidance system. Once Guidance Module GUI is launched, this button is switched off by default.

### 5.7.3 X-Plane® UDP Communication

Preliminary unit tests were performed while designing this tracking and guidance system using the MATLAB® simulation framework mentioned in previous section. However, in order to properly test and validate this guidance system it is necessary to make use of more realistic flight dynamics. For this reason is has been used a commercial flight simulator.

X-Plane® is a commercial flight simulator developed by Laminar Research [72]. This simulation software provides advanced communication features that allows user to create external plugins or add-on by making use of a standard communication protocol. In addition, X-Plane® provides to the user an interface to export to data files almost any parameter resultant from aircraft dynamics. This is aircraft position, heading, speeds and pitch-roll-yaw angles.

X-Plane® uses a standard communication protocol named User Datagram Protocol [5]. This protocol allows sending and receiving data to external software and its interaction with aircraft flight dynamics in runtime. It is known that UDP communication does not support advanced communication error detection and correction methods such as Automatic Repeat Request (ARQ) or Forward Error Correction (FEC) [22], which makes vulnerable for critical applications that requires these features. In this case, making use of ARINC communication standards is necessary. However, for this particular case, where a computer simulation is carried out, making use of UDP protocol has been considered enough.

4DT RS Guidance Module communication interface has been implemented as a stand-alone part by making use of UDP communication libraries developed by Amaro et al. [74].

Methods for sending and receiving datagrams and its future conversion to decimal base is provided by these libraries by making use of a structure named *UDP\_Pack*.

An *UDP\_Pack* is composed by ten (10) fields of *double* type as shown in the following Table 5-1.

**Table 5-1: *UPD\_Pack* specification [74]**

Field	Description	Units
AUTO_ALT	Autopilot panel Altitude Hold	Feet
AUTO_HDG	Autopilot Panel Heading Hold	Degrees
AUTO_SPD	Autopilot Panel IAS Hold	Knots
AUTO_VS	Autopilot Panel Vertical Speed Hold	Feet / Minute
Curr_ALT	Current Aircraft Altitude	Feet
Curr_LON	Current Aircraft Longitude (from GNSS)	Degrees

Curr_LAT	Current Aircraft Latitude (from GNSS)	Degrees
Curr_HDG	Current Aircraft Heading (from HDG)	Degrees
Curr_TSPD	Current Aircraft True Airspeed	Knots
Curr_SPD	Current Aircraft Indicated Airspeed	Knots

First four (4) fields are used to send data to aircraft autopilot control panel. Last six (6) fields are used to receive data from aircraft systems. Position related information (latitude and longitude) is received from GNSS aircraft systems.

# ***Chapter 6***

## **TEST AND RESULTS**

### **6.1 Introduction**

Activities carried out to test the 4DT generator and guidance system were focused on fuel, time and noise reduction. For this section, it has been prepared three complete testing cases to demonstrate the functionalities and capabilities of developed system.

In section 6.2, it is presented a test case based on fuel and time reduction for a full flight between Paris (LFPG) and Amsterdam (EHAM) airports.

Section 6.3 it is proposed a solution to a real noise problem around Gatwick airport vicinities by optimizing a section of the instrumental departure procedure of this airport.

Lastly, in section 6.4 it is evaluated a trajectory generated by the 4D trajectory generator by carrying out a flight simulation using the 4DT RS Tracking & Guidance module.

## 6.2 Test case (Fuel and Time Optimization)

**Flight Plan Selected:** Paris Charles de Guille (LFPG) to Amsterdam Schiphol (EHAM)

Paris - Amsterdam is an important route that covers a total distance of 313 miles. According to statistics released by Amsterdam Airport Report [50] in 2013, Paris-Charles de Gaulle is the third busiest route from Schiphol airport handling more than 1.1 million passengers per year. The top airlines demanding this route are Air France and KLM [50].

A standard flight plan between LFPG and EHAM has been composed by a total of nine (9) waypoints defined as shown in the following Table 6-1.

**Table 6-1: LFPG – EHAM Flight Plan**

Waypoint	Latitude (deg)	Longitude (deg)
LFPG	49.0097	2.5478
NURMO	49.8261	2.7553
PERON	49.9125	2.8400
CMB	50.2281	3.1515
VEKIN	50.4042	3.2750
ADUTO	50.5150	3.3617
FERDI	50.9126	3.6370
HELEN	51.2353	3.8690
EHAM	52.3081	4.7642

LFPG to EHAM is a high-traffic route, however most crossroads and congested areas are present between fixes CMB and FERDI. Also, holding patterns in the vicinities of these airports during approach and descending phases are often needed due to limited airport landing capacities. Optimizing this trajectory could prevent performing these holding patterns and could improve traffic flow around conflicting points.

The main idea of this testing case is to demonstrate and validate the capabilities of 4D Trajectories Research Software v0.1a developed in this project by

optimizing a full flight plan while reducing Fuel or Time. Also providing a predicted time of arrival for each route is the first step to perform four dimensional procedures by future Flight Management Systems (FMS) and ATM Ground Systems.

### 6.2.1 Setting Up Testing Case in 4DT RS

Paris Charles de Guille (LFPG) airport is composed by four runways heading to East-West direction (09-27 or 08-26). Runway 09R has been used for this testing case. Initial parameters were used for speed, mass and heading as shown in Table 6-2.

**Table 6-2:** Initial Conditions

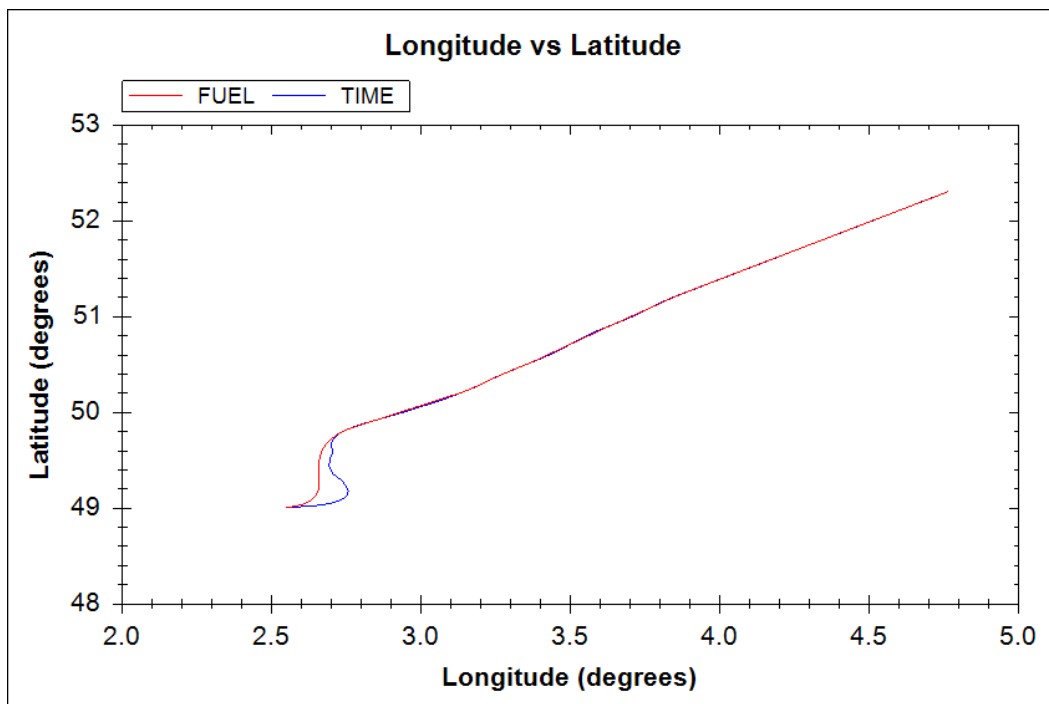
Parameter	Initial Value
Heading (deg)	090
Speed (knots)	180
Mass (lb)	185,000

Each waypoint has been input in the software using *Database Viewer Tool* and Fight Plan Editor method. It has been decided to run two different cases for same flight plan so it is possible to compare and analyse the difference between fuel and time optimized trajectories. A first case has been run using *Flight Time* as main performance index and another case has been setup to reduce *Fuel Consumption*. Total optimization CPU-time was around 4 minutes for each case using a 2.63 GHz dual-core processor and 8GB of RAM.

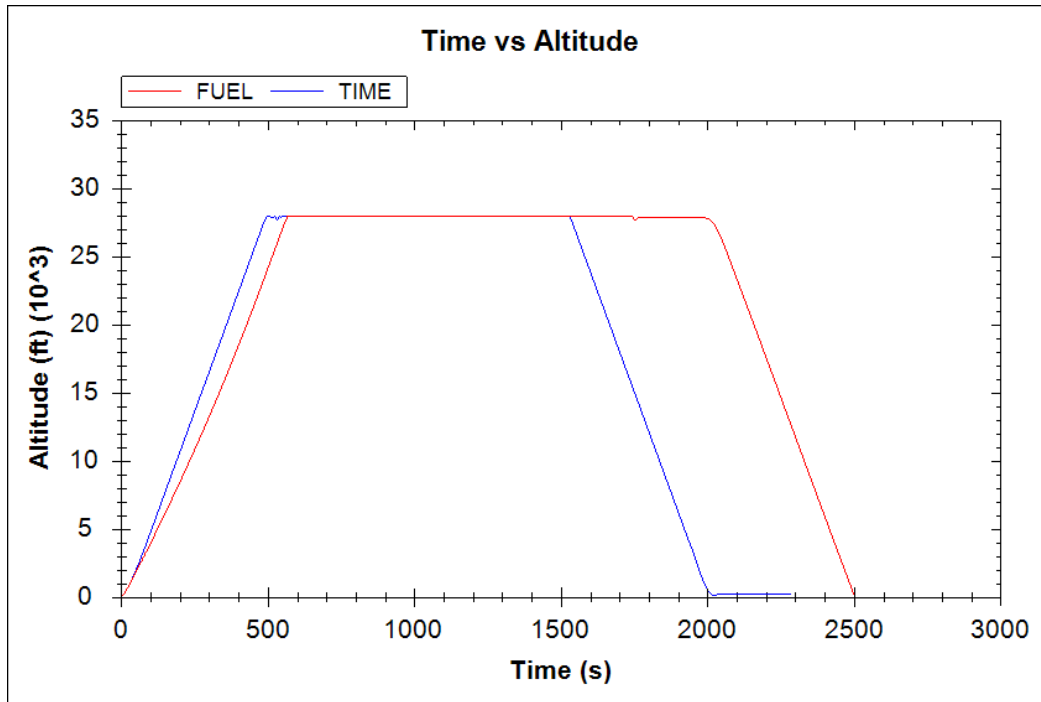
Results of both trajectories are shown and can be easily contrasted using the embedded map feature. Also vertical and horizontal profiles could be compared using embedded plots. Figure 6-1 shows the map view obtained from 4DT RS. Figure 6-2 and Figure 6-3 show the trajectory horizontal and vertical profiles.



**Figure 6-1: Fuel vs Time Optimization | Map View**



**Figure 6-2: Fuel vs Time Optimization | Horizontal Profile**



**Figure 6-3:** Fuel vs Time Optimization | Vertical Profile

Predicted time of arrival, TOC and BOD points have been exported and are shown in the following Table 6-3 and Table 6-4.

**Table 6-3:** TOC, BOD and PTA | Time Optimization

Waypoint	Latitude (deg)	Longitude (deg)	PTA (min)
LFPG	49.0097	2.5478	-
NURMO	49.8261	2.7553	8.15
<b>TOC</b>	<b>49.86</b>	<b>2.76</b>	<b>8.36</b>
PERON	49.9125	2.8400	8.94
CMB	50.2281	3.1515	13.07
VEKIN	50.4042	3.2750	15.09
ADUTO	50.5150	3.3617	16.32
FERDI	50.9126	3.6370	20.76
HELEN	51.2353	3.8690	24.43
<b>BOD</b>	<b>51.25</b>	<b>3.88</b>	<b>25.68</b>
EHAM	52.3081	4.7642	37.05

**PTA = Predicted time of arrival**



**Table 6-4: TOC, BOD and PTA | Fuel Optimization**

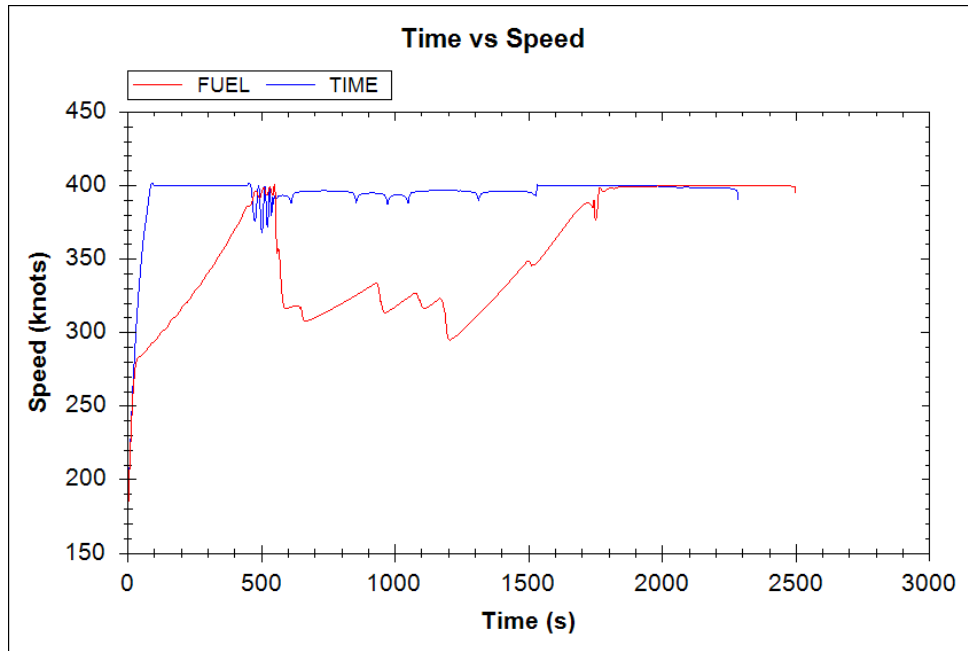
Waypoint	Latitude (deg)	Longitude (deg)	PTA (min)
LFPG	49.0097	2.5478	-
NURMO	49.8261	2.7553	8.45
<b>TOC</b>	<b>49.82</b>	<b>2.75</b>	<b>9.14</b>
PERON	49.9125	2.8400	9.27
CMB	50.2281	3.1515	14.33
VEKIN	50.4042	3.2750	16.83
ADUTO	50.5150	3.3617	18.34
FERDI	50.9126	3.6370	23.99
HELEN	51.2353	3.8690	27.92
<b>BOD</b>	<b>51.62</b>	<b>4.19</b>	<b>33.6</b>
EHAM	52.3081	4.7642	40.62

**PTA = Predicted time of arrival**

In one hand, aircraft speed has been maximal for almost whole trajectory while reducing flight time. In the other hand, fuel reduction case results shown an average of 337.2 knots which is equivalent to 1% slower than Time case (390.2 knots).

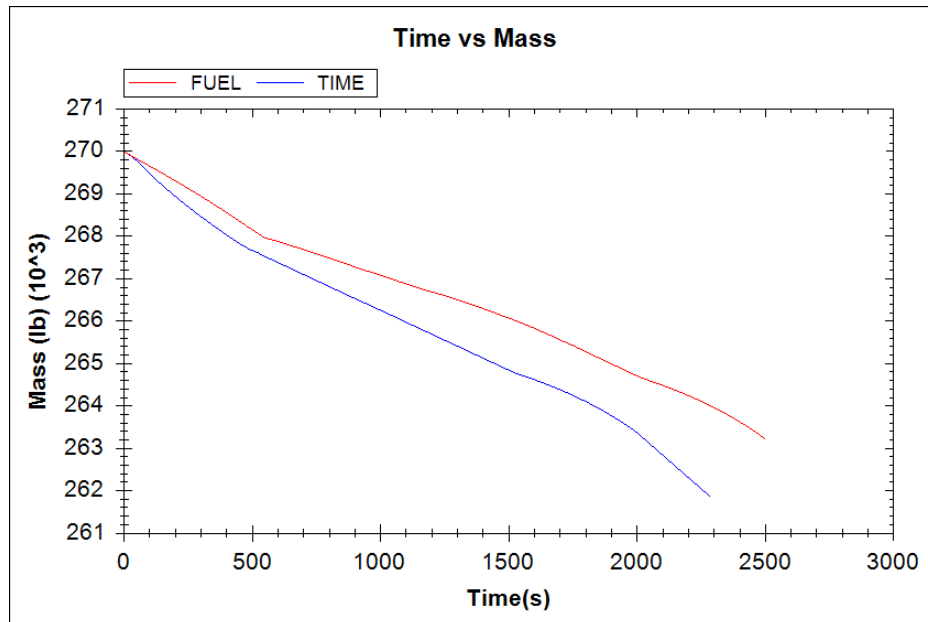
Predicted time of arrival (PTA) has been calculated based on fly-by waypoints. For these particular cases, predicted time of arrival shows a fast ascending phase. Top of Climb (TOC) point is obtained around 8-10 minutes after take-off.

Descending phase evaluation for both cases is different than ascending phases comparison. Results of Figure 6-3 show remarkable continuous descent paths. However, descent trajectory has been extended for 13 minutes on Time optimization results while Fuel optimization results shows a faster descending phase.



**Figure 6-4: Fuel vs Time Optimization | Speed Profile**

Fuel consumption has been reduced in 16% with respect to Time optimization case. In contrast, flight time has been reduced 8.5% with respect to Fuel optimization case. Figure 6-5 shows mass reduction comparison between both trajectories.



**Figure 6-5: Fuel vs Time Optimization | Fuel Consumption**

*Detailed View Tool* has been used to obtain aircraft status at each optimization node along the flight plan. It can be considered as a useful tool to follow aircraft trajectory and detecting possible errors in optimization process.

Different flight data values regarding to flight cost and flight cost per hour can be obtained by changing price per hour parameter in *Aircraft Editor*. Flight Information using a standard Jet-A1 fuel price at Paris Charles De Guille published in September 2015 Fuel Survey [63], is shown in the following Table 6-5.

**Table 6-5:** Flight Information | Time and Fuel Optimization

Parameter	Trajectory-F	Trajectory-T
<b>Total Time (min)</b>	<b>41.64</b>	<b>38.06</b>
Total Distance (nautical miles)	273.48	273.48
Fuel Consumption Rate (lb / hr)	9,761.51	12,811.83
Fuel Consumption Rate (gal / m)	3.7	4.44
<b>Fuel Consumed (gal)</b>	<b>1,011.04</b>	<b>1213.13</b>
Fuel Consumed (lb)	6,774	8,128
Fuel Cost per hour (usd)	4,370.83	1,986.64
Fuel Cost (usd)	3,033.13	1,261.4

**Trajectory-T** = trajectory resultant from time optimization

**Trajectory-F** = trajectory results from fuel optimization

### 6.3 Test case (Noise Optimization)

**Airport Selected:** London Gatwick Airport (EGKK)

London Gatwick airport is the UK second largest airport and the busiest single-runway international airport of the world servicing over 31 million passengers in 2010. In order to demonstrate the capabilities of 4DT Generator Research Software, a Noise Optimization test has been run in order to solve a potential problem related to Gatwick's Standard Departure Procedures.

This testing case is based on information provided in Flight Evaluation Report released in 2010 [35] by Gatwick Airport (EGKK) Flight Evaluation Unit (FEU).

Gatwick Airport is equipped with noise monitors situated exactly 6.5km from the roll point of the runways. Noise levels are registered at each monitor when aircraft take-off. If noise registered is greater than noise limits fixed by Department of Transport, a noise infringement is registered.

Despite of no infringements were registered for any of total 120,249 departures registered in 2010, the number of enquiries registered in that year raised to its maximum value since 2006. A total of 6,936 enquiries from 409 callers have been registered [35]. Aircraft noise, low flying and night flights are some of common causes of these enquiries. Also, it could be inferred that low flying and night flights complaints could be also related to aircraft noise. An important fact is that biggest amount of these enquiries were registered in populated areas located more than 10 miles away from Gatwick Airport where noise optimization procedures can be applied.

Hever, Marsh Green and Edenbridge are villages located around 12-13 miles East of Gatwick Airport's runway. Around 56% of all enquiries were registered in these villages, which is equivalent to more than 1000 enquiries from each area. Other villages such as Lingfield, Dorking and Crawley registered less than 300 enquiries. Table 6-6 shows most important enquiries registered in 2010.

**Table 6-6:** Enquiries registered in 2010 [35]

Village	Enquiries	Callers
East Gringstead	1,378	13
<b>Marsh Green</b>	<b>721</b>	<b>2</b>
<b>Hever</b>	<b>568</b>	<b>4</b>
<b>Edenbridge</b>	<b>459</b>	<b>29</b>
Lingfield	331	8
Dorking	132	25
Crawley	73	50

Current Noise Abatement procedures forbid overflying Horley and Crawley villages located at North and South of airport respectively. However, the location of more affected areas mentioned above matches with trajectories defined by Gatwick Standard Instrumental Departure (SID) procedures for exits CLN 5P/5W, DVR 2P/2W and BIG 3P/3W. Marsh Green, Hever and Edenbridge registered a total of 1748 enquiries. Note that East Gringstead does not match with this SID therefore it has been neglected for this case.

There are three different departure procedures that head to East or North-East exit waypoints: CLACTON (CLN), DOVER (DVR) and BIGGIN (BIG). The main problem occurs between airport and intermediary fix TUNBY (N51 10.1 E00 19.5) located at 14 miles off airport. Navigation charts inform to maintain a 4% minimum climb gradient until 3000ft, however no procedures exist in order to avoid flying over affected populated areas in these aircraft routes towards TUNBY fix.

The mentioned three SID procedures have been mixed in just one figure. The problem exists in the red section which connects Gatwick Airport to TUNBY fix. The affected villages have been highlighted in order to show the problem.

# BIGGIN, CLACTON, DOVER



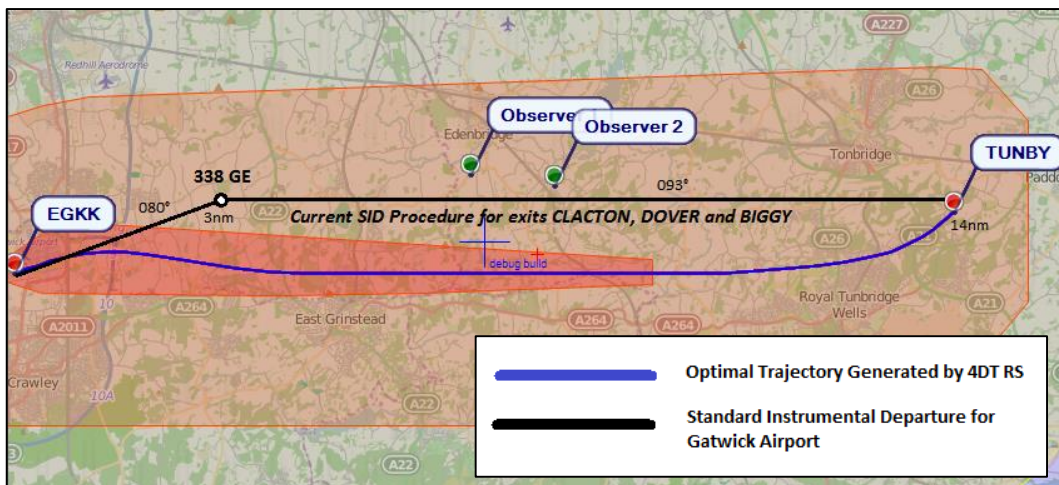
### 6.3.1 Setting Up Testing Case in 4DT RS

The main idea of this case is to show how it is possible to optimize aircraft trajectory in order to reduce noise levels on departure phase with respect to affected villages.

Two noise observers were positioned in the testing scenario. One of them was located at Marsh Green village boarder and the other was located at Hever village border. The idea is to optimize the trajectory around these two observers. Initial waypoint is located at runway 08R/26L and final waypoint is located at TUNBY fix (N51 10.1, E00 19.5, 6000 ft).

Initial heading has been set 80 (Runway 08R) and approximate value for aircraft V2 speed has been set as initial speed. Also, aircraft limits has been set to obtain speed values no greater than 250 knots due to airspace limitations for flight level below 10.000 ft.

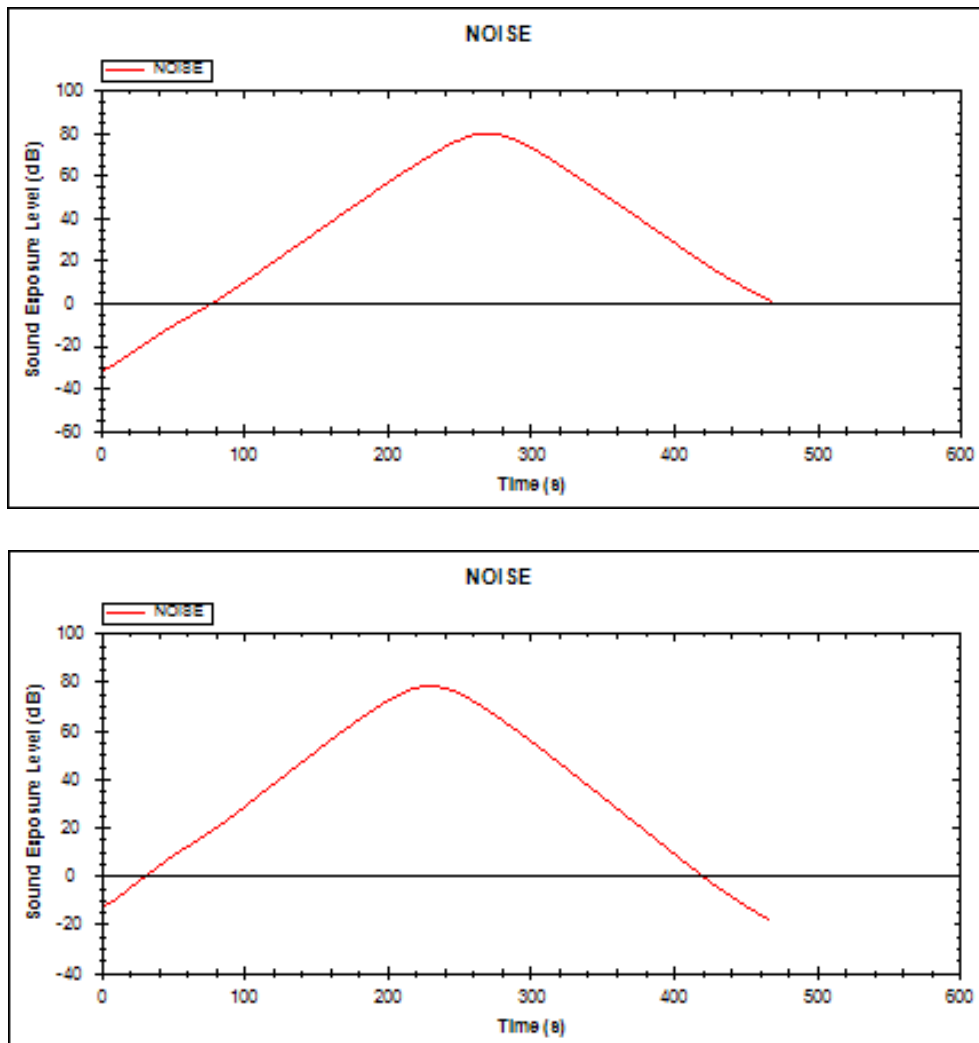
Testing case has been run using noise performance index and 50% of balance between observer 1 and observer 2 (This means there is not special preference to optimize with respect to one of the observers). The following figure shows the testing case screenshot run in 4DT Trajectory Research Software. Noise level tool has been used to draw a contour plot composed by four noise levels of 30 (dark red), 45 (orange), 65 (yellow) and 90 dB. Engine noise model used for this case was PW2036 at maximum thrust.



**Figure 6-6:** Optimal trajectory compared to SID baseline procedures

Figure 6-6 above shows optimal trajectory generated by 4DT Research Software (blue) contrasted to current SID baseline procedures for exits DVN, CLN and BIG (black).

Optimization results show an important change in aircraft trajectory with respect to current SID. Noise levels perceived at Observers 1 and 2 are shown in Figure 6-7.



**Figure 6-7:** Noise Exposure Level at Observer 1 (top) and Observer 2 (bottom)

Maximum noise levels perceived at observers are around 78 dB. It is important to highlight that these results show most pessimist scenario. In order to obtain a more likely scenario, it should be considered that these observers were located at the border of affected villages (Marsh Green and Hever). In addition, it has

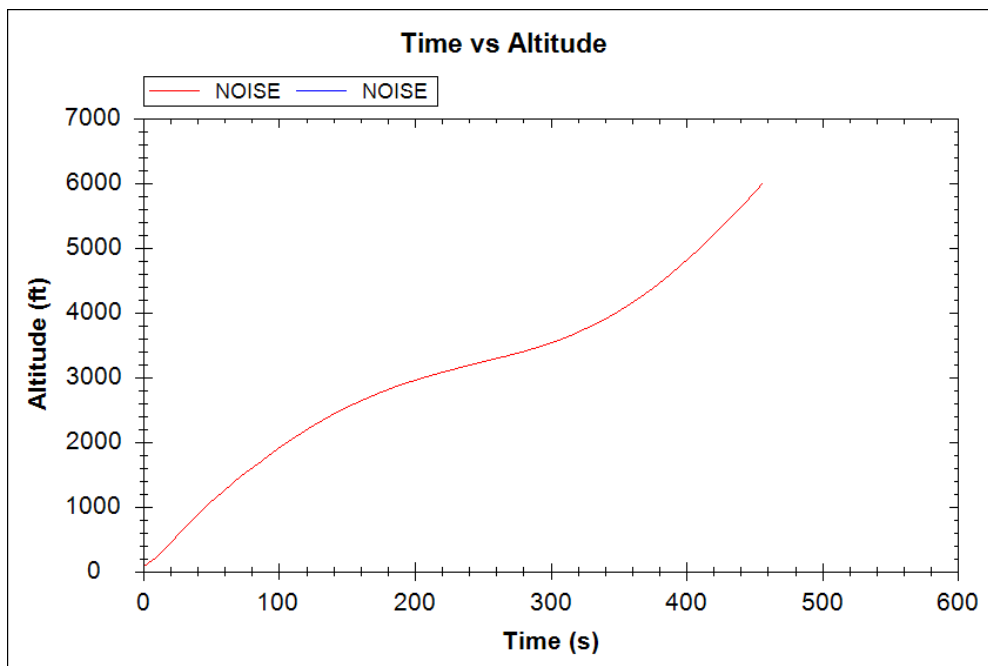


been considered that engines produce always maximum possible noise (max. thrust) during optimization process, however, trajectory results shows that thrust average was 88,262 lb (54.75% of total thrust). This means that actually maximum sound levels perceived at populated areas of these villages would be considerable lower in most departure operations.

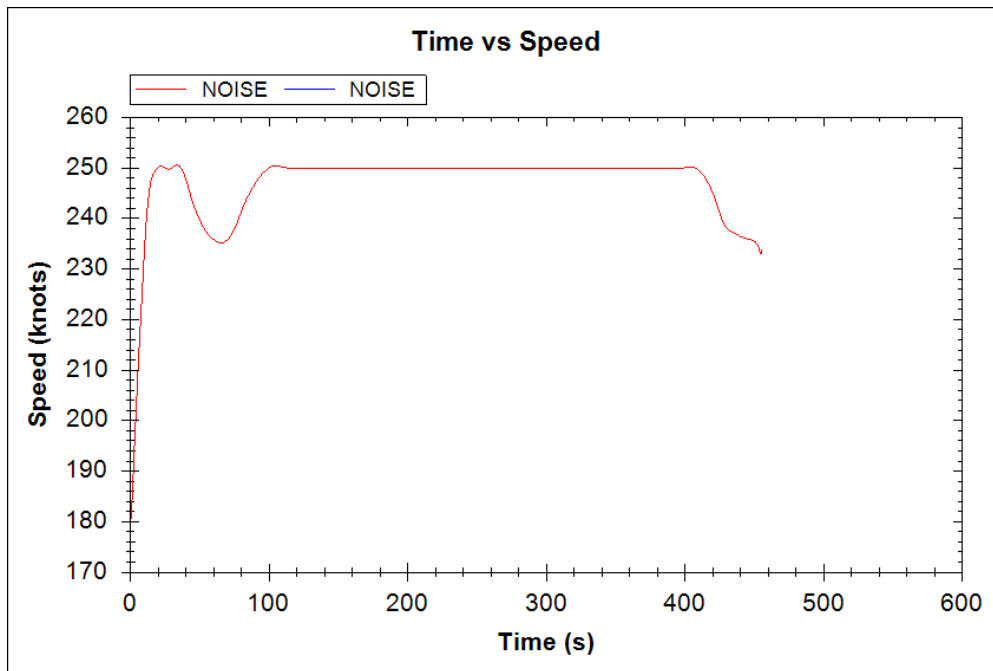
Current SID procedures strategy is that aircraft follows runway 08R heading (080) for 3 nautical miles up to 338 GE point and then heading changes to 92 degrees until arriving to TUMBY fix. Optimal trajectory shows a significant difference with respect to these SID procedures. Aircraft heading changed from the very beginning to avoid flying close to affected villages. TUMBY is intersected with a final heading of 62 degrees which could be beneficial for North-East exits (CLACTON and BIGGIN).

Altitude profile is defined by changes in vertical speed, especially between 3000ft to 4000ft where vertical speed changes from 400 ft/min to 1000 ft/min. 6000 ft. are reached at 6.42 min which result in an average of 940 ft/min which is equivalent to a climb gradient no greater than 3%.

Altitude profile and speed profile are shown in Figure 6-8 and Figure 6-9.



**Figure 6-8: Altitude Profile – Noise Optimization**



**Figure 6-9:** Speed Profile – Noise Optimization

## **6.4 Test case (Trajectory evaluation using 4DT RS T&G Module)**

A testing case has been created based on the reference trajectory obtained from Section 6.2 in order to test and evaluate generated trajectory. This is a full flight from Paris Charles de Guille airport (LFPG) to Amsterdam Schiphol airport (EHAM) exported using 4D Trajectories Research Software.

This testing case has been designed in order to test the following aspects:

1. Test and validate trajectories generated by 4DT Research Software by carrying out a full flight along the trajectory.
2. Test the aircraft response capability to return to reference trajectory when for any reason it loses its normal course.
3. Test tracking system capabilities when computing lateral and vertical errors.
4. Test guidance system capabilities to calculate guidance commands that allows aircraft follow a reference trajectory.
5. Test Tracking & Guidance module graphic user interface and communication interface.
6. Test guidance indicators proposed for Primary Flight Display (PFD) and existent indicators for Navigation Display (ND) when setting up values in the autopilot control panel.

Initial setup of Tracking & Guidance module was performed before carrying out this test. It has been exported the trajectory generated by 4DT RS and imported into the 4D RS Guidance Module.

It has been configured X-Plane® in order to allow interaction with external software following procedures explained in its documentation [72]. Subsequently, it has been tested the sending/receiving functions of guidance module.

It has been created a new flight simulation environment using a standard jet aircraft with similar characteristics that the one used to generate the trajectory.

Additionally, weather parameters have been setup as shown in the following Table 6-7.

**Table 6-7:** Visibility and Cloud configuration

<b>Cloud Layer 1 Altitude (ft)</b>	10,000
<b>Cloud Layer 2 Altitude (ft)</b>	20,000
<b>Cloud Layer 3 Altitude (ft)</b>	26,000
<b>Visibility (miles)</b>	12.5

For this test case, also three (3) wind layers<sup>2</sup> has been setup as shown in following Table 6-8.

**Table 6-8:** Wind speed configuration

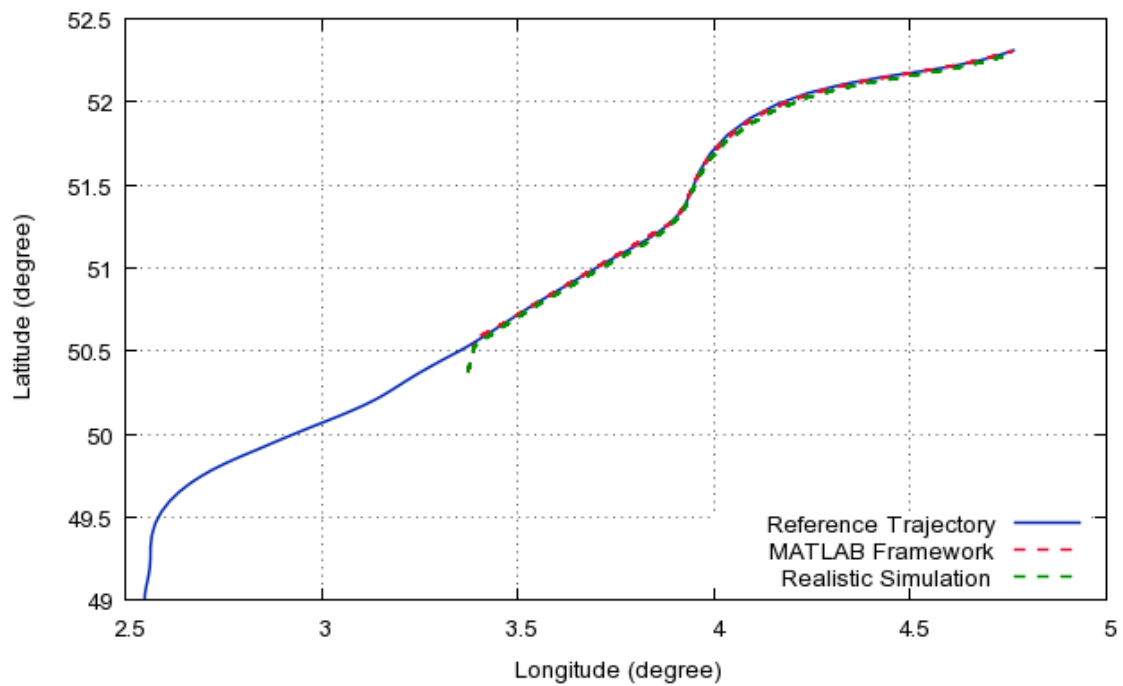
<b>Wind Layer Altitude (ft)</b>	<b>Wind Speed (kts)</b>
5,000	12
7,500	21
25,000	18

Aircraft initial position has been set to 7 nautical miles off and 4,000 feet below a randomly-selected point located at middle of trajectory. In this way, it has been possible to measure the reaction capacity of tracking and guidance system to return aircraft to reference course and track.

Figure 6-10 shows aircraft initial position in horizontal plane, reference trajectory and aircraft trajectory followed once tracking & guidance module has been activated. Additionally, it has been added a simulation result of MATLAB® simulation framework generated for lateral guidance unit testing.

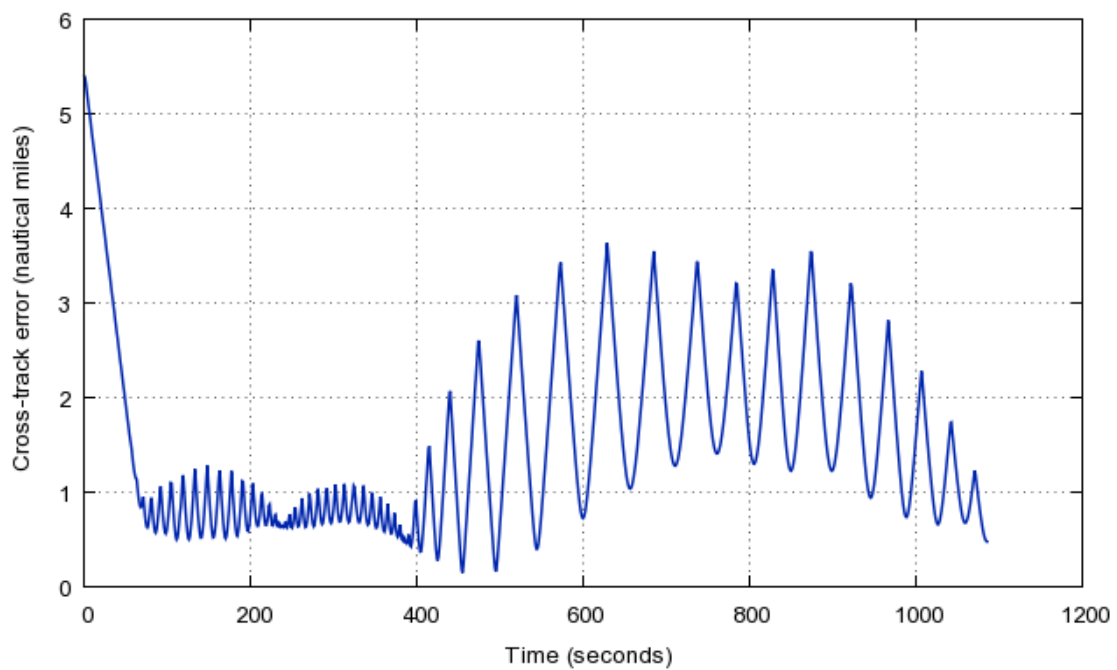
---

<sup>2</sup> X-Plane® v9.7 weather setup limits the number of available wind layers to three (3)



**Figure 6-10:** Horizontal Profile | Tracking & Guidance Module

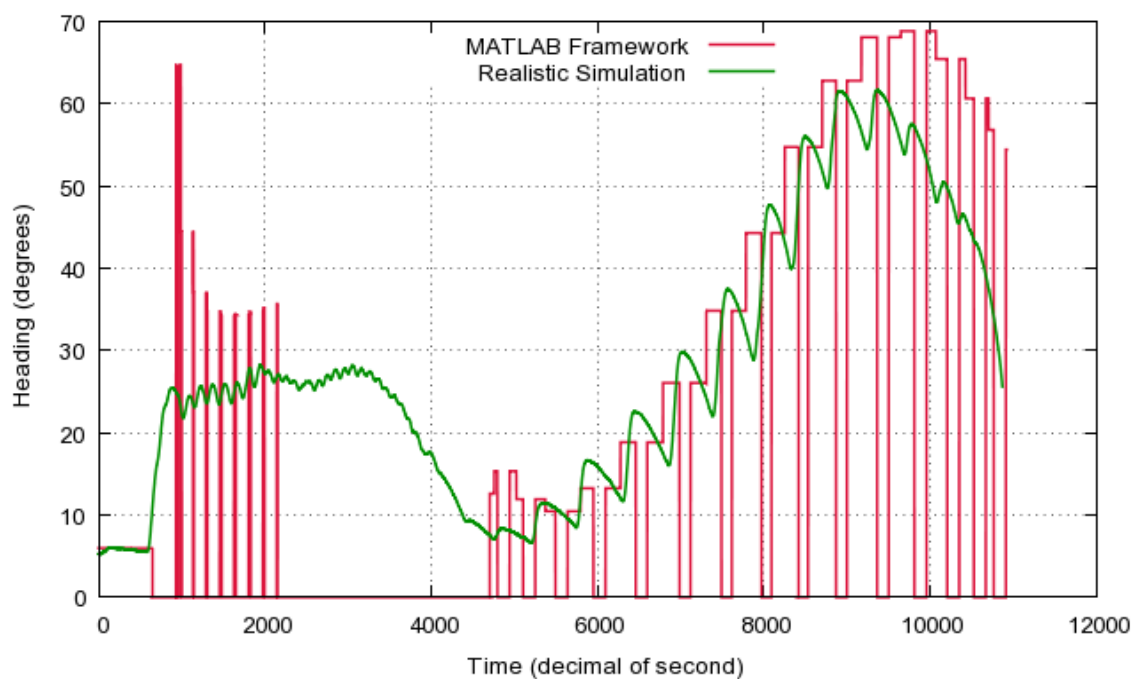
Once reference trajectory was intercepted, guidance commands allowed aircraft to follow its course along this trajectory until reaching last waypoint (EHAM).



**Figure 6-11:** Cross-track error for lateral guidance

Cross-track error is shown in Figure 6-11. It is visible that error maintains below 1.5 nm until 400 seconds. It has been detected that algorithm for guidance is sensitive to arc shaped trajectories that require a constant computation of new heading. This is evidenced after 400 seconds when cross-track error oscillates around 4 nautical miles.

Figure 6-12 shows that aircraft heading obtained from X-Plane® realistic simulation also matches similar values than obtained using MATLAB® Simulation Framework.



**Figure 6-12: Aircraft heading comparison**

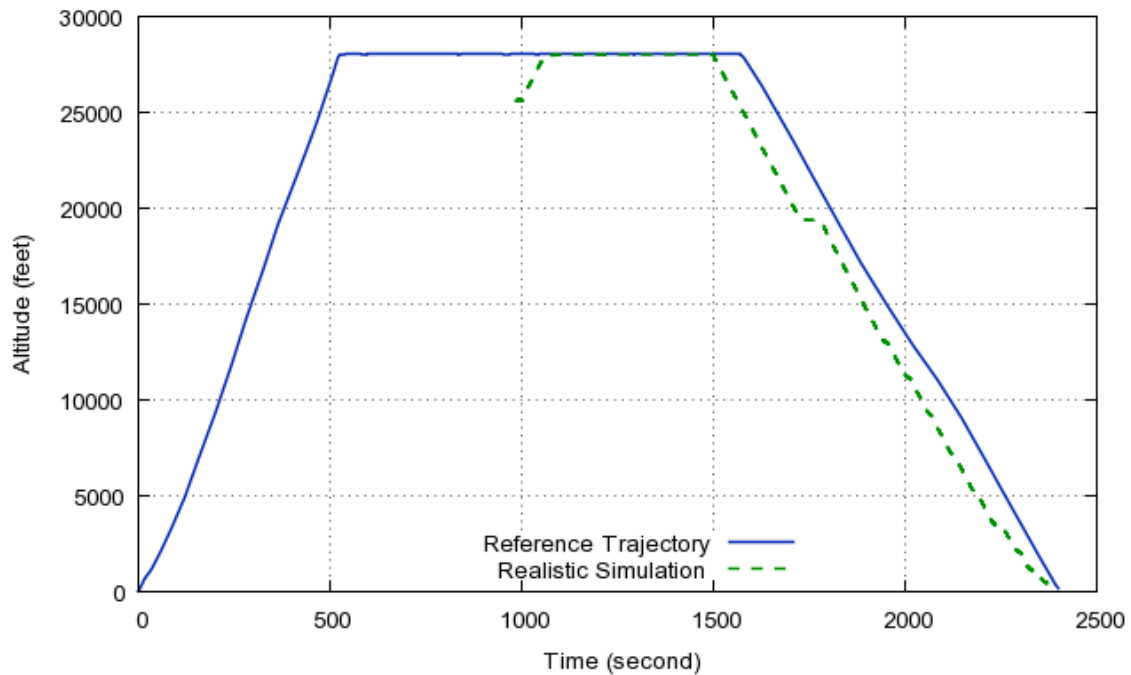
Figure 6-13 shows aircraft vertical profile obtained from X-Plane® simulation contrasted with Reference Trajectory. It is shown the difference of 4,000 feet below reference path for initial aircraft position.

First seconds of simulation, guidance system allows aircraft to increase vertical speed in order to reduce altitude error. Cruise altitude is maintained until descend phase.

Descent is started some seconds before by aircraft. This effect is produced because of reference trajectory information is taken with respect to far in advance

by guidance system. Reference node for vertical guidance has been implemented to be always the following node of cross-track error point.

Autopilot control is quite effective and interprets guidance commands with no-delay. Despite of descending phase has been started aircraft trajectory tends to maintain similar descending rate than reference path.



**Figure 6-13:** Vertical Profile | Tracking & Guidance Module

Guidance visual indicators on Attitude Indicator in Primary Flight Display moved correctly according to direction of reference trajectory. For descending phase, horizontal bar maintained a position below center line which means that aircraft required to expedite its descending rate. This effect has been produced due to same reasons explained before, reference node detected was always taken far in advance with respect to aircraft current position.

Non-smoothness has been detected at some points for *Time* optimization results in tests explained in Section 6.2. However, no special effort has been experienced by aircraft/autopilot system to follow the trajectory around these points. Nevertheless, since this is a particular case where non-smoothness detected has not affected aircraft course, an exhaustive analysis of the

optimization framework and NLP might be required in order to understand the cause of these non-smooth points in order to prevent undesired effects of this issues in future tests.

As mentioned in previous chapter, SNOPT solver has been properly linked to 4DT core, however no tests have been carried out yet using this NLP solver.

## **6.5 Known Limits and Improvements**

Current 4DT generator and guidance system runs properly. Important efforts have been carried out in order to create a stable version. However, since this is a first version some non-smoothness effects has been detected while finding an optimal solution. Some errors and bugs have been identified in testing milestone; this section pretends the user to be aware of them.

The system aims at predicting almost any trajectory defined by almost any flight plan. However, it is known that computer resources are limited. Some actions have been taken in order to adjust the number of nodes (and consequently memory) allocated by PSOPT. These adjustments are totally dependent of the number of waypoints. Even thought, according to tests performed, flight plans composed by more than 25-30 waypoints produce a general application crash due to a lack of memory.

Since Top of Climb (TOC) point coincides with first waypoint, the position of first waypoint is important. The optimal control solver could diverge while finding an optimal solution if cruise altitude is set to be reached too early. It is important that input flight plan information be as much as consistent as possible with equations (4-11), (4-12), (4-20) and (4-21).

Despite of database contains a considerable amount of entries (150,000+), it is known that this database should be improved with more significant data by containing more nav aids or arrival/departure procedures.

It is known that convex hull-based algorithms are an efficient way to produce polygons based on a set of points. Their application in noise grid computation is valid and works properly. However, using algorithms that produce concave



polygons would be more appropriate in order to obtain more accurate noise contour plots. Nevertheless, concave polygon algorithms are known for their extreme complexity and unlike convex solutions, they do not produce an exclusive solution. For this reason, making use of them has been neglected in this project.

It is known that 4DT Core could take a considerable amount of time to generate a trajectory when a medium-large flight plan is input. This effect has been detected in Windows platform. It has not carried out a formal testing case aiming at comparing the performance for different platforms, however a priori it is possible to detect that execution time on Linux systems is considerable shorter than Windows systems.

By decreasing the number of nodes to be used by PSOPT it has been possible to reduce the amount of execution time. However, it is known that reliability and time execution efficiency are important factors to take into account for on-board systems. For this reason, it is considered that execution time could be a potential improvement for future versions.

# Chapter 7

## CONCLUSIONS AND FUTURE WORK

### 7.1 Conclusion

The main objective of this project was developing a 4D Trajectories Generator and Guidance System.

This project resulted on developing *4D Trajectories Research Software v0.1*. A Software Suite that involves a 4D Trajectories Generator, a set of tools to evaluate and compare optimal trajectories and a Tracking & Guidance system that can be used to validate and test these trajectories by using a commercial flight simulator. Based on the experiences obtained while developing and testing this system and contrasting with proposed objective, it is possible to drawn the following conclusions:

An extensive review of current Flight Management Systems (FMS) and Flight Management Computer (FMC) functionalities including Required Time of Arrival (RTA) features has been carried out. This review phase compared different options existing in the current market. It has been determined that there exists current features that could be enhanced in order to fulfil the new requirements proposed by future Avionics and Air Traffic Management (ATM) systems.

4D Trajectories Generator has been designed, developed and included into the core of 4D Trajectories Research Software. This trajectory *synthesizer* allows user to obtain optimal trajectories in order to reduce time, fuel or noise while providing

an easy method for configuring, testing and comparing generated trajectories. The trajectory generator assembles the data input by the user, which is mainly composed by aircraft performance data, flight plan and initial conditions to subsequent generate an optimal trajectory. This is achieved by setting up an Optimal Control Problem (OCP) and using PSOPT solver to find a solution, if exists.

4D Trajectories Research Software also provides a set of tools embedded into a user-friendly graphic interface that allows user to analyse, compare and setup testing cases. Detailed view, navigation database, noise contours and multi-trajectory plotting are some of features including in this software. Most optimal control solver preferences can be configured using the graphic user interface. Lastly, all these tools are properly explicated into the integrated *Help* form, which also includes several examples about how to use this software.

4D Tracking & Guidance system has been developed in order to test and validate optimal trajectories developed by 4DT Generator. It has been integrated as a 4DTR module. Aircraft dynamics data for Tracking & Guidance module is obtained from a commercial flight simulator that is connected to external applications by using a User Datagram Protocol (UDP).

Numerous tests has been carried out in order to check and validate the capabilities of 4D Trajectories Research Software including all its modules.

A particular case has been based on a high-traffic European route between Paris and Amsterdam in order to test and validate the trajectory generation capabilities of the software. This test has been focused based on reducing fuel consumption and flight time.

Noise reduction tests have been carried out based on a real problem which is affecting vicinities of London Gatwick airport. This test resulted into a proposal to modify part of standard instrumental departure based on resultant noise optimal trajectory.

Predicted trajectory of first testing case has been validated by simulating this flight using the Tracking & Guidance module. Results shown that aircraft can follow the trajectory with an error below 4 nautical miles.

Finally, it has been analysed the existence of possible improvements that could be taken into account for future works of this project.

## **7.2 Future Work**

This project has been completed based on objectives proposed. However, it has been identified several features or improvement that could be achieved in the future.

Performance computation and execution time seems to be one of the most critical applications for future development. It has been proved that 4DT Core executes faster in Linux-based systems. Developing a Graphic User Interface (GUI) for this operating system would extend the domain of usability of 4DT RS. However, on-board applications or embedded systems requiring real time capabilities, it would be necessary to significantly improve computation time by optimizing the code execution.

A wind model has been neglected for this trajectory generator. Despite performance computation and execution time could be affected by the use of a more complex model. The wind effect in fuel consumption and time of arrival is important to improve the accuracy of the solution. Therefore it is suggested to improve this synthesizer by including a wind model with components in translational axis.

Synthesizing optimal trajectories based on noise reduction is limited to departure phases including two reference points. Noise data available reflects aircraft engine noise, and it is demonstrated that other aircraft components have to be taken into account (e.g. flaps, slats, landing gear) for arrival phases. It is considered that including noise reduction capabilities for arrival phases including more than two observers could be a challenging and interesting work for the future.

It is considered that success on finding an optimal trajectory depends of how smart-enough is the optimal control setup. However, the presence of non-smooth points have been detected in some solutions. It is suspected that this could be matter of the non-linear programming solver (IPOPT). As mentioned before, SNOPT is linked to PSOPT, however no tests has been carried out using this NLP solver.

Database included in this software is limited to a set of most-used nav aids (FIX, VOR, NDB) and airports. Future work could turn around improving the components included in database by adding capabilities to support STAR, SID, Jet/Victor airways and any of path terminators explained in ARINC-424 specification.

### **7.3 Challenges**

It has been spent one year in development of this project. The journey from concept to 4DT Generator & Guidance System has been challenging and exciting. Current version works properly and it is close to be a computational stable version. Some of challenges experienced while developed this project are explained in this section.

It has been put a considerable amount of time and effort into studying control theory and finding the best way to use an optimal control solver into a stand-alone application. After performing numerous tests using DYNOPT, MATLAB® *fmincon*, IPOPT and other solvers, it has been decided the best option for this project was making use of PSOPT libraries.

Installing and using PSOPT libraries was exceptionally challenging in the first phase of the project. Continuous communication with its developer (Dr. Victor Becerra) and collaborators (Dr. Markus Sauermann) has carried out in order to successfully compile these libraries.

As typical in aerospace field, deficiency of information in public domain has been another challenge of this project. Obtaining access to BADA files or real flight data to compare costs, fuel consumption and validate tests has input few delays in the project. Most of these issues were solved with extra-time worked.

Finally, a considerable amount of time has been put in carrying out numerous tests by the author and other collaborators. Fortunately, several bugs and improvements have been obtained from these tests and contributed to obtain more stables versions.

# REFERENCES

- [1] V. Boltyanskii, R. Gamkrelidze, L. Pontryagin. (1956). К теории оптимальных процессов [Towards a Theory of Optimal Processes]. Dokl. Akad. Nauk SSSR (in Russian) 110 (1): 7–10.
- [2] L. Pontryagin (1962), “Mathematical Theory of Optimal Processes”. New York: John Wiley & Sons.
- [3] M. Guelman (1971). “A qualitative study of proportional navigation.” IEEE - Transactions on Aerospace and Electronic Systems, Vol. 7, No. 4, 637-643.
- [4] A. Bryson, Y. Ho (1975), “Applied Optimal Control”, John Wiley & Sons, New York, NY.
- [5] D. Reed. “RF-768 User Datagram Protocol” Internet Standard. (1980). [Online] [Last Visited 17/09/15] <https://tools.ietf.org/html/rfc768>
- [6] P. Preparata, M. Shamos. “Convex-hull: Basic Algorithms”. Springer New York. (1985).
- [7] J. Logsdon, L. Biegler (1989), “Accurate solution of differential-algebraic optimization problems”, Chem. Eng. Sci., (28):1628 - 1639.
- [8] M. Smith (1989). “Aircraft Noise”. Cambridge University Press.
- [9] O. Stryk, R. Bulish (1992). “Direct and Indirect Methods for Trajectory Optimization”. Annals of Operations Research. Vol 37. 357-373. [Online] [Last Visited 17/09/15] <https://personalrobotics.ri.cmu.edu/files/courses/papers/Stryk92-optimization.pdf>
- [10] B. Etkin, L. Reid (1995), “Dynamics of Flight: Stability and Control”, 3rd edition, Wiley.
- [11] G. Elnagar, M. Kazemi, M. Razzaghi (1995). “The Pseudospectral Legendre Method for Discretizing Optimal Control Problems”. IEEE Transactions on automatic Control, Vol 40. No 10, October 1995.

- [12] S. Ghawghawe, D. Ghose (1996). "Pure proportional navigation against time-varying target maneuvers." IEEE - Transactions on Aerospace and Electronic Systems, Vol 32, No. 4, 1336-1347, October 1996
- [13] N. Shneydor (1998). "Missile Guidance & Pursuit: Kinematics, Dynamics and Control." Woodhead Publishing. First Edition. January, 1998.
- [14] "DO236B. Minimum Aviation System Performance Standards: Required Navigation Performance for Area Navigation". Radio Technical Commission for Aeronautics (RTCA). October, 2003.
- [15] T. Prevot, V. Battiste, E. Palmer, S. Shelder (2003). "Air Traffic Concept Utilizing 4D Trajectories and Airborne Separation Distance". American Institute of Aeronautics and Astronautics (AIAA). [Online] [Last Visited 09/17/15]  
[http://humanfactors.arc.nasa.gov/ihh/DAG\\_WEB/public/publications/AIAA-2003-5770-GNC.pdf](http://humanfactors.arc.nasa.gov/ihh/DAG_WEB/public/publications/AIAA-2003-5770-GNC.pdf)
- [16] V. Becerra (2004) "Solving optimal control problems with state constraints using nonlinear programming and simulation tools". IEEE Transactions on Education, 47(3):377-384.
- [17] B. Geiger, J. Horn, A. DeLullo, L. Long, A. Niessner (2006). "Optimal Path Planning of UAVs Using Direct Collocation with Nonlinear Programming". GNC Conference at American Institute of Aeronautics and Astronautics.
- [18] A. Herndon, M. Cramer, K. Sprong, R. Mayer (2007). "Analysis of Advanced Flight Management Systems (FMS), Flight Management Computer (FMC) Field Observations Trials, Vertical Path". IEEE-26<sup>th</sup> Digital Avionics Systems Conference
- [19] D. Smedt, G. Berz (2007). "Study of the Required Time of Arrival function of current FMS in an ATM Context". IEEE and 26<sup>th</sup> Digital Avionics Systems Conference.
- [20] D. Hull (2007), "Fundamentals of Airplane Flight Mechanics", Springer.
- [21] J. Canino, L. Deniz, J. Garcia, J. Besada and J. Casar (2007). "A Model to 4D Descent Trajectory Guidance". Institute of Electrical and Electronic Engineers (IEEE).



- [22] M. Mörz. (2007) "Analog Signal Processing in Forward Error Correction (FEC) Decoders".
- [23] "NextGen Avionics Roadmap Version 1.0," Joint Planning and Development Office, 24 October 2008.
- [24] A. Herndon, M. Cramer, K. Sprong (2008). "Analysis of Advanced Flight Management Systems (FMS), Flight Management Computer (FMC) Field Observations Trials, Radius-To-Fix Path Terminators". 27<sup>th</sup> Digital Avionics Systems Conference.
- [25] A. Smith (2008) "Proportional Navigation with Adaptive Terminal Guidance for Aircraft Rendezvous", Journal of Guidance, Control, and Dynamics, Vol. 31, No. 6, pp. 1832-1836.
- [26] "Air Traffic NextGen Briefing" Federal Aviation Administration (FAA). (2009). [Online] [Last Visited 25/03/15] [http://www.faa.gov/air\\_traffic/briefing/](http://www.faa.gov/air_traffic/briefing/)
- [27] J. Klooster, A. Amo, P. Manzi, (2009). "Controlled Time-of-Arrival Flight Trials". Eighth USA/Europe Air Traffic Management Research and Development Seminar.
- [28] R. Mall (2009). "Fundamentals of Software Engineering". 3<sup>rd</sup> Edition. PHI Learning Private Limited.
- [29] A. Rao, (2009). "Survey of Numerical Methods for Optimal Control," 2009 AAS/AIAA Astrodynamics Specialist Conference, AAS Paper 09-334, Pittsburgh, PA.
- [30] "Arrival Manager (AMAN): Implementation Guidelines and Lesson Learned". EUROCONTROL. Edition Number 0.1. December, 2010.
- [31] C. Sinclair (2014), "Honeywell Next Generation Flight Management System Meets New Air Traffic Requirements". Honeywell Aerospace Media Center (2010). [Online] [Last Visited 26/01/14] <http://www51.honeywell.com/honeywell/news-events/press-releasesdetails/03.04.10HONNextGenNewAirTrafficRequirements.html>
- [32] K. Chircop, M. Xuereb, D. Zammit, E. Cachia, (2010). "A Generic Framework for Multi-Parameter Optimization Of Flight Trajectories". University of Malta and Cranfield University. ICAS 2010.

- [33] T. Yamasaki, S. Balakrishnan, (2010). "Sliding Mode Based Pure Pursuit Guidance for UAV Rendezvous and Chase with A Cooperative Aircraft". American Control Conference (ACC), 5544-5549.
- [34] "787-8 Flight Crew Operations Manual". The Boeing Company (2007-2010).
- [35] "Flight Performance Team 2011 Annual Report". Gatwick Airport. [Online] [Last Visited 09/15/15] [http://www.gatwickairport.com/globalassets/publicationfiles/business\\_and\\_community/all\\_public\\_publications/2011/flight\\_performance\\_team\\_2011\\_annual\\_report.pdf](http://www.gatwickairport.com/globalassets/publicationfiles/business_and_community/all_public_publications/2011/flight_performance_team_2011_annual_report.pdf)
- [36] D. Naidu (2003). "Optimal Control Systems". CRC Press.
- [37] J. Betts (2011), "Practical Methods for Optimal Control Using Nonlinear Programming". Advances in Design and Control. SIAM. PP 85-87.
- [38] "SESAR Factsheet: I-4D - Flying a new dimension" Single European Sky Research Team (SESAR). (2012). N° 01/2012.
- [39] A. Herndon (2012). "Flight Management Computer (FMC) Navigation Database Capacity". Integrated Communication Navigation and Surveillance (ICNS) Conference.
- [40] B. Tian, Q. Zong (2012). "3DOF Ascent Phase Trajectory Optimization for Aircraft Based on Adaptive Gauss Pseudospectral Method". Third International Conference on Intelligent Control and Information Processing.
- [41] J. Anderson (2012), "Introduction to Flight", University of Maryland, Mc. Graw Hill.
- [42] M. Xuereb, K. Chircop, D. Zammit (2012). "GATAC – A Generic Framework for Multi-Parameter Optimization of Flight Trajectories". AIAA Modeling and Simulation Technologies Conference. Minneapolis, Minnesota, USA.
- [43] O. Winter, H. de Campos, V. Carruba, (2012). "ASTER: A Brazilian Mission to an asteroid". Asteroids Comets Meteors.
- [44] S. Vaddi, G. Sweriduk, M. Tandale (2012). "4D Green Trajectory Design for Terminal Area Operations using Non-linear Optimization Techniques".

- Guidance, Navigation and Control Conference. Minneapolis, Minnesota, USA.
- [45] S. Vaddi, G. Sweriduk, M. Tandale (2012). "Design and Evaluation of Guidance Algorithms for 4D-Trajectory-Based Terminal Airspace Operations". 12th AIAA Aviation Technology Integration and Operations (ATIO) Conference.
- [46] T. Gruning, A. Rauh, H. Aschemann (2012). "Feedforward control design for a four-rotor UAV using direct and indirect methods" Methods and Models in Automation and Robotics (MMAR), 2012 17th International Conference.
- [47] "20-year Forecast of Annual Number of IFR Flights (2012 - 2035)". EUROCONTROL. (2013).
- [48] Federation of American Scientist (2013) "AFPAM 10-1403" [Online] [Last Visited 26/03/15] <http://www.fas.org/man/dod-101/usaf/docs/afpam10-1403.htm>
- [49] K. Bousson, P. Machado (2013). "4D Trajectory Generation and Tracking for Waypoint-Based Aerial Navigation", WSEAS Transactions on Systems and Control, 3 (vol. 8), pp. 105-119.
- [50] "Traffic Review 2013". Schiphol Amsterdam Airport. [Online] [Last Visited 29/03/15] <http://trafficreview2013.schipholmagazines.nl/traffic-review-2013.pdf>
- [51] Y. Diaz, S. Lee, M. Egerstedt, S. Young, (2013). "Optimal Trajectory Generation for Next Generation Flight Management Systems", 32nd Digital Avionics Systems Conference.
- [52] "Historical Foundation for ATM Research". NASA – Aviation Systems Divisions (2014). [Online] [Last Visited 09/16/15] <http://www.aviationsystemsdivision.arc.nasa.gov/research/foundations/index.shtml>
- [53] A. Gardi, R. Sabatini, S. Ramasamy, T. Kistan (2014). "Real-Time Trajectory Optimization Models for Next Generation Air Traffic Management Systems", Applied Mechanics and Materials, vol. 629, pp. 327-332, Trans Tech Publications.

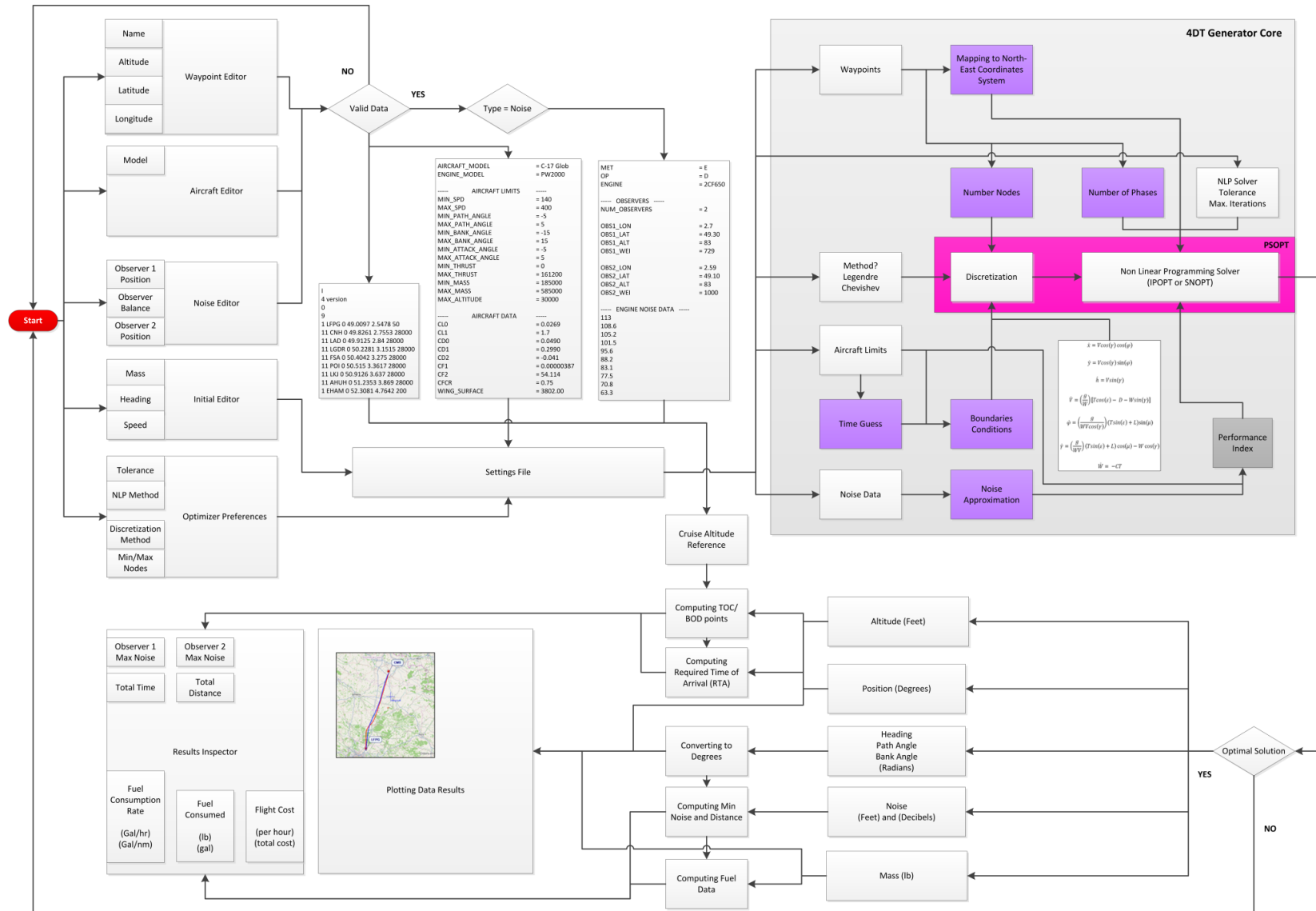
- [54] D. Ghose (2014). "Guidance of Missiles". National Program me on Technology Enhanced Learning (NPTEL).
- [55] M. Soler, B. Zou, M. Hansen (2014). "Flight trajectory design in the presence of contrails: Application to a multiphase mixed-integer optimal control approach", *Transportation Research Part C*, 48, pp. 172-194.
- [56] S. Han, H. Bang (2014). "Proportional Navigation-Based Optimal Colission Avoidance for UAV". *International Conference of Autonomous Robots and Agents*.
- [57] "AIM-9 Sidewinder". Federal of American Scientists (FAS). [Online] [Last Visited 17/09/15] <http://fas.org/man/dod-101/sys/missile/aim-9.htm>
- [58] "ARINC 424 Navigational Data". Jeppensen Sanderson. [Online] [Last Visited 09/17/15] <http://ww1.jeppesen.com/industry-solutions/aviation/government/arinc-424-navigational-data-service.jsp>
- [59] "ARINC 424 Specification". EUROCONTROL [Online] [Last Visited 09/17/15] <http://www.eurocontrol.int/articles/arinc424-specification>
- [60] "CleanSky – About" [Online] [Last Visited 17/09/15] [www.cleansky.eu](http://www.cleansky.eu)
- [61] "CTAS Tools Gallery". NASA – Aviation Systems Divisions. [Online] [Last Visited 09/09/15] <http://www.aviationsystemsdivision.arc.nasa.gov/multimedia/ctas/index.shtml>
- [62] "DYNOPT". M. Fikar, M. Čížniar. Department of Information Engineering and Process Control, Faculty of Chemical and Food Technology, Slovak University of Technology in Bratislava. [Online] [Last Visited 26/03/15] <http://www.kirp.chtf.stuba.sk/moodle/mod/page/view.php?id=5460>
- [63] "Fuel Price Survey - September 2015". European, Middle East & African Aviation Fuel Price Survey. [Online] [Last Visited 17/09/15] <http://tmdg.co.uk/misc/fuel.php>
- [64] "GreatMaps Libraries for .NET Framework". Published at CodePlex <https://greatmaps.codeplex.com/>
- [65] "Instrument Procedures Handbook". Federal Aviation Administration (FAA). (2014). [Online] [Last Visited 17/09/15]

- [http://www.faa.gov/regulations\\_policies/handbooks\\_manuals/aviation/instrument\\_procedures\\_handbook/media/FAA-H-8083-16.pdf](http://www.faa.gov/regulations_policies/handbooks_manuals/aviation/instrument_procedures_handbook/media/FAA-H-8083-16.pdf)
- [66] “Integrated Noise Model”. Federal Aviation Administration (FAA). [Online] [Last Visited on 09/21/15] [https://www.faa.gov/about/office\\_org/headquarters\\_offices/apl/research/models/inm\\_model/](https://www.faa.gov/about/office_org/headquarters_offices/apl/research/models/inm_model/)
- [67] “IPOPT Project” [Online] [Last Visited 26/03/15] <https://projects.coin-or.org/Ipopt>
- [68] “Microsoft .NET Home Website”. Microsoft Corporation. [Online][Last Visited 09/20/15] <http://www.microsoft.com/net>
- [69] “Microsoft HTML Help 1.4”. Windows Dev Center, Microsoft Corporation. [Online] [Last Visited 09/20/15] [https://msdn.microsoft.com/en-us/library/windows/desktop/ms670169\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms670169(v=vs.85).aspx)
- [70] “PSOPT – Pseudo Spectral Optimization Solver”. V. Becerra. [Online] [Last Visited 26/03/15] <http://www.psopt.org>
- [71] “SNOPT – Stanford Business Software Inc.” [Online] [Last Visited 26/03/15] <http://www.sbsi-sol-optimize.com/>
- [72] “X-Plane Documentation”. Laminar Research. [Online] [Last Visited 17/09/15] [http://wiki.x-plane.com/Chapter\\_5:\\_X-Plane\\_Menus#Data\\_Input\\_.26\\_Output](http://wiki.x-plane.com/Chapter_5:_X-Plane_Menus#Data_Input_.26_Output)
- [73] “ZedGraph Libraries”. Published at SourceForge under LGPLv2 License. <http://zedgraph.sourceforge.net/samples.html>
- [74] M. Amaro, C. Barrado, D. Rudinskas (2015). “Design of a flight management system to support four-dimensional trajectories.” Aviation Tailor & Francis Vol 19. 58-65.
- [75] “Aerospace Trajectory Optimization Software | Products”. ASTOS Solutions. [Online] [Last Visited 01/10/15]. <https://www.astos.de/>
- [76] “General Mission Analysis Tool (GMAT)”. National Aeronautics and Space Administration (NASA). [Online] [Last Visited 01/10/15] <https://gmatsfc.nasa.gov/>

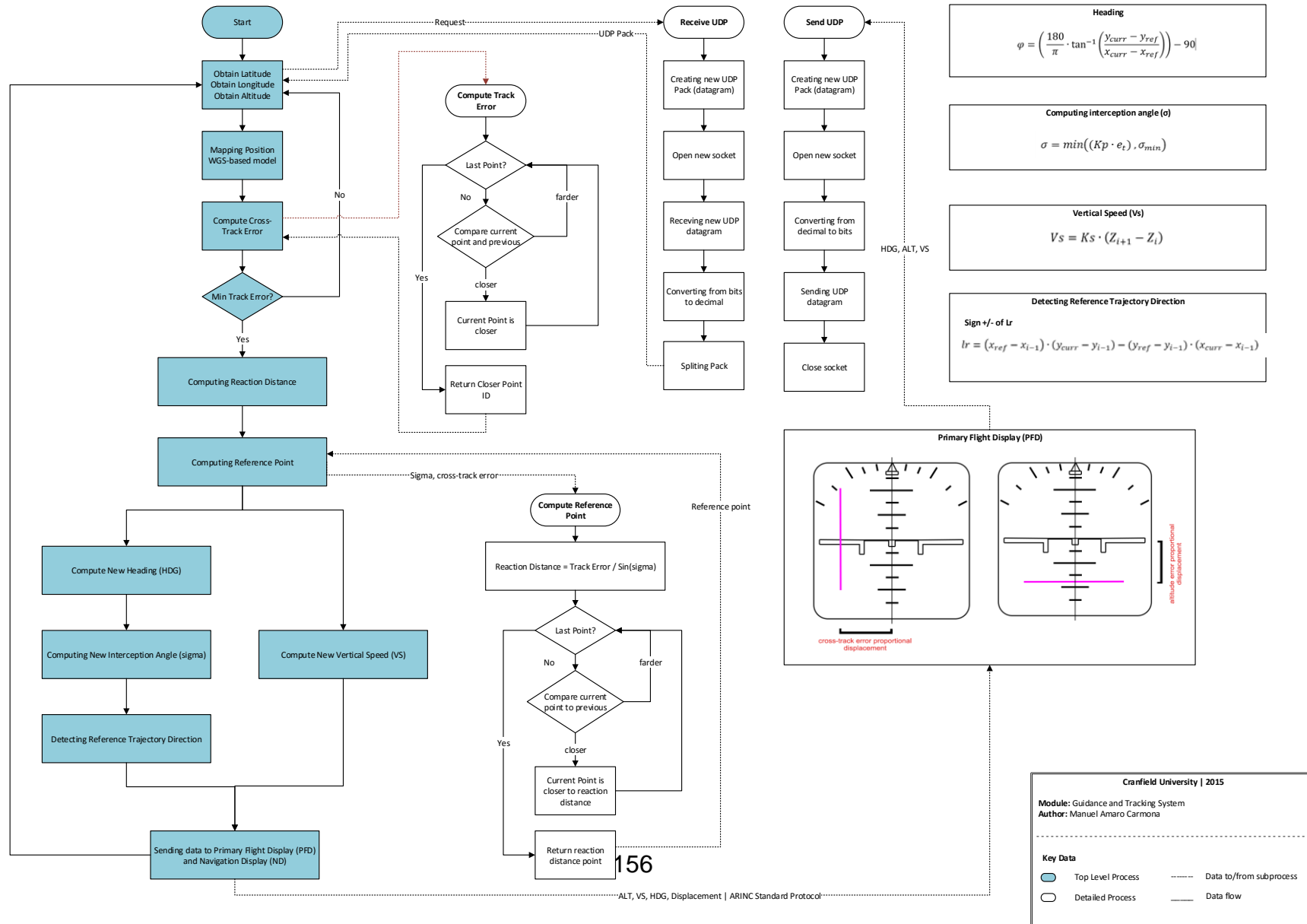
- [77] “Optimal Trajectories by Implicit Simulation (OTIS)”. Charles Hargraves, Steve Paris. Boeing Company & NASA. [Online] [Last Visited 01/10/15] <https://otis.grc.nasa.gov/authors.html>
- [78] “User Manual for Base of Aircraft Data (BADA)”, EUROCONTROL Experiment Center, Bretigny, France, 2004

# APPENDICES

## Appendix A 4DT RS Overview



# Appendix B Tracking & Guidance System Overview





## Appendix C 4DT RS Functions

**Function:** MappingWGS

**Inputs:** latitude1, longitude1, latitude2, longitude2

**Output:** flat\_point

- 1) Create a flat point
- 2) Compute delta values  
 $\text{delta\_lat} = \text{lat2} - \text{lat1}$   
 $\text{delta\_lon} = \text{lon2} - \text{lon1}$
- 3) Compute Latitude and Longitude in Feet  
 $\text{north\_distance} = \text{delta\_lat} * \text{KM2FT}(110.54)$   
 $\text{east\_distance} = \text{delta\_lon} * \text{KM2FT}(111.32) * \cos(\text{DEG2RAD}(\text{lon\_avg}))$
- 4) End

**Description:** Computing Time Constraints

```
// Computing End Time Lower
double EndTime_lower = pow((pow((flat_waypoints[i].east_dist - flat_waypoints[i-1].east_dist), 2.0) + pow((flat_waypoints[i].north_dist - flat_waypoints[i-1].north_dist), 2.0)), 0.5) / (velocity_upper[i-1]);

// Computing End Time Upper
double EndTime_upper = pow((pow((flat_waypoints[i].east_dist - flat_waypoints[i-1].east_dist), 2.0) + pow((flat_waypoints[i].north_dist - flat_waypoints[i-1].north_dist), 2.0)), 0.5) / (velocity_lower[i-1]);
```

**Function:** integand\_cost

**Inputs:** control, states, dot\_states

**Output:** Cost

- 1) If (Type = "Time")  
 $\text{Cost} = (\text{velocity} - \text{velocity\_max})^2 / \text{velocity\_max} + \dots (\text{altitude} - \text{ref\_altitude})^2 / \text{altitude\_max};$
- 2) If (Type = "Noise")  
 $\text{Cost} = \text{Total\_SEL} + (\text{altitude} - \text{ref\_altitude})^2 / \text{altitude\_max};$
- 3) If (Type = "Fuel")  
 $\text{Cost} = (\text{mass} - \text{initial\_mass}) + (\text{altitude} - \text{ref\_altitude})^2 / \text{altitude\_max};$

**Function:** compute\_noise

**Inputs:** noise\_vector, distance\_vector, observer\_distance

**Output:** SEL

```
1) if (observer_distance <= 4000)

    1.1) while (i < m)

        1.1.1) Compute vector index
                far = m - i - 1;
                near = i;

        1.1.2) Compute partial A1 and B1 values
                B1 = (Noise[near]*log10(Dist[far]) -
Noise[far]*log10(Dist[near]))
                ... / (log10(Dist[far]) - log10(Dist[near]));
                A1 = (Noise[near] - B1) / log10(Dist[near]);

        1.1.3) Compute A1 and B1 counters
                A_count = A_count + A1;
                B_count = B_count + B1;

    end while

    1.2) Compute A1 and B1 final values
        A1 = A_count / m;
        B1 = B_count / m;

    1.3) Compute noise value
        noise = A*log10(R) + B;

2) else

    2.1) Compute slope
        slope = (noise_at_25000 - noise_at_4000) / (25000 - 4000);

    2.2) Compute slope
        noise = slope * (R - 4000) + noise_data.noise4000;

1) End
```

<b>Function:</b> PlotWaypoints <b>Inputs:</b> n/a <b>Output:</b> n/a
<ol style="list-style-type: none"> <li>1) Read data from Flight Plan file  List&lt;waypoint&gt; waypoints = ReadFlightPlanDataGrid();</li> <li>2) For each waypoint <ol style="list-style-type: none"> <li>2.1) Create point  new PointLatLng(point.lat, point.lon);</li> <li>2.2) Create marker  new GMarkerGoogle(route_waypoint, ... new Bitmap(path_icons);</li> <li>2.3) Add marker to layer list  markersOverlay.Markers.Add(marker);</li> </ol> </li> <li>3) Add layer to map  gMapControl1.Overlays.Add(markersOverlay);</li> <li>2) End</li> </ol>

<b>Function:</b> PlotTrajectory <b>Inputs:</b> n/a <b>Output:</b> n/a
<ol style="list-style-type: none"> <li>1) Read data from trajectory files  route = ReadTrajectoryFile();</li> <li>2) Create route  GMapRoute Grout = new GMapRoute(route, "Trajectory");</li> <li>3) Adding route to layer list  routes.Routes.Add(Grout);</li> <li>4) Add layer to map  gMapControl1.Overlays.Add(routes);</li> <li>3) End</li> </ol>

**Function:** FillNoiseGrid

**Inputs:** noise\_grid, max\_limit, min\_limit

**Output:** list\_points\_db

- 1) For each point in noise grid
  - 1.1) If ( min\_limit < SEL < max\_limit )
    - 1.1.1) Create new point  
PointLatLng noise\_waypoint = new PointLatLng(point.lat, point.lon);
    - 1.1.2) Add point to a noise list  
list\_points\_db.Add(noise\_waypoint);
    - 1.1.3) Create new marker  
new GMarkerGoogle(noise\_waypoint, grid\_icon);
    - 1.1.4) Add marker to layer  
NoiseGridOverlay.Markers.Add(marker);
- 2) End

**Function:** ComputeFlightData

**Inputs:** latitude, longitude, mass

**Output:** flight\_data

- 1) Create new FlightData object  
new FlightData();
- 2) Computing orthodomic distance  
delta\_lon = Math.Abs(lon[0] - lon[lon.Count - 1]);  
delta\_lat = Math.Abs(lat[0] - lat[lat.Count - 1]);  
  
north\_dist = delta\_lat \* KM2FT(110.54);  
east\_dist = delta\_lon \* KM2FT(111.32) \* Math.Cos(DEG2RAD(delta\_lon));
- 3) Compute Total distance in miles  
flight\_data.distance = Math.Sqrt(Math.Pow(east\_dist, 2) +  
... Math.Pow(north\_dist, 2)) \* 0.00018939;
- 4) Computing Final Mass  
flight\_data.fuel\_consumed\_lb = mass.Max() - mass.Min();  
flight\_data.fuel\_consumed\_gal = flight\_data.fuel\_consumed\_lb / 6.7;  
flight\_data.fuel\_cost = flight\_data.fuel\_consumed\_gal \* fuel\_price\_gal;
- 5) Computing Fuel Consumed and Price

```

flight_data.total_time = time.Max() / 60;
flight_data.fuel_consumed_hour = flight_data.fuel_consumed_lb * 60 /
    ... flight_data.total_time;
flight_data.fuel_consumed_mile = flight_data.fuel_consumed_gal /
    ... flight_data.distance;
flight_data.price_hour_flight = flight_data.fuel_cost * 60 /
    ... flight_data.total_time;

```

6) End

**Function:** ComputeFlightData

**Inputs:** latitude, longitude, mass

**Output:** flight\_data

- 1) Create new list  
`List<double> TimeOfArrival = new List<double>();`
- 2) For each aircraft\_position
  - 2.1) if distance\_to\_waypoint < 0.1
    - 2.1.1) Add time of arrival to list  
`TimeOfArrival.Add(time_of_arrival);`
- 3) End

**Function:** ComputeBODTOCPoints (Part 1)

**Inputs:** latitude, longitude, time, cruise\_alt

**Output:** TOC

- 1) For each altitude
  - 1.1) Obtain TOC point index  
`Math.Abs(altitude[i] - cruise_alt) < 50`
- 2) Obtain time at TOC point  
`TOC_lat = latitude[i];`  
`TOC_lon = longitude[i];`
- 3) End Part 1

**Function:** ComputeBODTOCPoints (Part 2)

**Inputs:** latitude, longitude, time, cruise\_alt

**Output:** BOD

- 1) For each altitude (start at point where first part stopped)
  - 4.1) Obtain BOD point index  
`altitude[j] < cruise_alt - 400`
- 2) Obtain time at TOC point  
`BOD_lat = latitude[i];`  
`BOD_lon = longitude[i];`
- 3) End

**Function:** FindNearestPoint

**Inputs:** cursor\_position

**Output:** n/a

- 1) Find nearest point  
`Find(IsNear(point.Lat, point.Lng));`
- 2) Finding index of nearest point  
`index = FindIndexInPositionVector(found);`
- 3) Plotting the point  
`new GMarkerGoogle(found, new Bitmap(path_icon);`
- 4) Retrieve point information (altitude, position, heading...)
- 5) Add marker to layer  
`nearpoints_marker.Markers.Add(marker);`
- 6) End

## Appendix D MATLAB® Simulation Framework

```
% -----  
% Lateral Guidance System Simulation Test (Simple version)  
% Created by Manuel Amaro  
% Cranfield University | 2014 - 2015  
% -----  
  
clear all  
  
% Loading data from 4DT_RS  
time = load('Dataset/time.dat');  
speed = load('Dataset/velocity.dat');  
lat = load('Dataset/lat.dat');  
lon = load('Dataset/lon.dat');  
alt = load('Dataset/altitude.dat');  
lon_avg = abs(lon(1) - lon(length(lon)));  
  
lat_simulator = load('Dataset/lat_sim.txt');  
lon_simulator = load('Dataset/lon_sim.txt');  
hdg_simulator = load('Dataset/hdg_sim.txt');  
alt_simulator = load('Dataset/alt_sim.txt');  
crosserror_simulator = load('Dataset/crosstrack_error.txt');  
  
% Simulation parameters  
simtime_prev = 1; % Used to record previous  
simulation time  
total_simulation_time = 300000; % Total simulation time  
(maximum)  
reaction_time = 130; % Reaction time in seconds  
sigma = 30; % Angle at which aircraft  
intercepts reference path in degrees  
RNP = 1; % Required Navigation  
Performance parameter in nautical miles  
  
% Initial conditions  
aircraft_hdg = 90; % Heading in Degrees  
aircraft_lat = 50.37876; % Latitude in Degrees  
aircraft_lon = 3.37269; % Longitude in Degrees  
aircraft_TAS = 49; % Aircraft speed  
aircraft_alt = 20000;  
  
% Mapping 4DT_RS data to WGS model  
for t = 1:length(lat)  
    [lon_nm(t), lat_nm(t)] = MappingWGS(lat(1), lon(1), lat(t), lon(t),  
    lon_avg);  
end  
  
% Creating a standardized time vector  
for i=1:length(alt_simulator)  
    time_general(i) = i;  
end  
  
% Mapping WGS model and converting aircraft position to nautical miles  
[aircraft_lon_nm, aircraft_lat_nm] = MappingWGS(lat(1), lon(1),  
aircraft_lat, aircraft_lon, lon_avg);
```

```

% Computing error and obtaining error index
track_error_nm = inf;
for i = 1:length(lat_nm)
    if(sqrt((lat_nm(i) - aircraft_lat_nm)^2 + (lon_nm(i) -
aircraft_lon_nm)^2) < track_error_nm)
        track_error_nm = sqrt((lat_nm(i) - aircraft_lat_nm)^2 +
(lon_nm(i) - aircraft_lon_nm)^2);
        track_error_index = i;
    end
end

% Uncomment to test
% hold on;
% plot(lon(node_index), lat_nm(node_index), 'o')
% plot(lon_nm(track_error_index), lat_nm(track_error_index), 'o')
% plot(lon_nm, lat_nm);

for sim_time = 1 : total_simulation_time

    if(track_error_nm > RNP)

        % Computing reaction distance

        % Based on a fixed reaction time
        % reaction_dist = KTS2KTSS(aircraft_TAS) * reaction_time;

        % Based on cross-track error
        reaction_dist = track_error_nm / sind(sigma);

        % Calculating node at which distance is closer to reaction
distance
        react_dist_tolerance = inf;
        for i = 1:length(lat_nm)
            if((abs(sqrt((lat_nm(i) - aircraft_lat_nm)^2 +
(lon_nm(i) - aircraft_lon_nm)^2) - reaction_dist) <
react_dist_tolerance) && i > track_error_index)
                react_dist_tolerance = abs(sqrt((lat_nm(i) -
aircraft_lat_nm)^2 + (lon_nm(i) - aircraft_lon_nm)^2) -
reaction_dist);
                node_index = i;
            end
        end

        % Uncomment to test
        % hold on;
        % plot(lon_nm(track_error_index), lat_nm(track_error_index), 'x')
        % plot(lon_nm(node_index), lat_nm(node_index), 'o')
        % plot(lon_nm(track_error_index), lat_nm(track_error_index), 'o')
        % plot(lon_nm, lat_nm);

        % Computing new aircraft heading
        m = (aircraft_lat_nm - lat_nm(node_index)) / (aircraft_lon_nm -
lon_nm(node_index));
        aircraft_hdg = -(57.29 * atan(m) - 90);
        aircraft_hdg_vec(sim_time) = aircraft_hdg;
    end
end

```



```

        % Moving aircraft
        aircraft_lat_nm = aircraft_lat_nm + (KTS2KTSS(aircraft_TAS) *
        cosd(aircraft_hdg));
        aircraft_lon_nm = aircraft_lon_nm + (KTS2KTSS(aircraft_TAS) *
        sind(aircraft_hdg));

        % Mapping aircraft position back
        [aircraft_lon, aircraft_lat] = MappingWGS_back(lat(1), lon(1),
        aircraft_lat_nm, aircraft_lon_nm, lon_avg);

        % Moving reference point (Proportional navigation typical
        characteristic)
        if(abs(simtime_prev - sim_time) > 10)
            node_index = node_index + 1;
            if(node_index > length(lat))
                node_index = length(lat);
            end
            simtime_prev = sim_time;
        end

        % Computing error and obtaining error index
        track_error_nm = inf;
        for i = 1:length(lat_nm)
            if(sqrt((lat_nm(i) - aircraft_lat_nm)^2 + (lon_nm(i) -
            aircraft_lon_nm)^2) < track_error_nm)
                track_error_nm = sqrt((lat_nm(i) - aircraft_lat_nm)^2
            + (lon_nm(i) - aircraft_lon_nm)^2);
                track_error_index = i;
            end
        end
        track_error_vec(sim_time) = track_error_nm;

        % Stop if aircraft is at final point
        if(track_error_index == length(lat))
            break;
        end

        % Plotting aircraft movement
        aircraft_lon_vec(sim_time) = aircraft_lon;
        aircraft_lat_vec(sim_time) = aircraft_lat;

    end

    % Plotting results
    figure;
    grid on;
    hold on;

    plot(lon, lat, 'b', 'LineWidth',2);
    plot(aircraft_lon_vec, aircraft_lat_vec, 'r--', 'LineWidth',2);
    plot(lon_simulator, lat_simulator, 'g--', 'LineWidth',2);

    figure;
    %plot(track_error_vec);
    hold on;
    plot(crosserror_simulator, 'g');

```

```
figure;  
hold on;  
time_mov = time_general/7.6998 + time(88);  
plot(time_mov, alt_simulator, 'g');  
plot(time, alt, 'b');  
  
figure;  
hold on;  
plot(aircraft_hdg_vec, 'r--', 'LineWidth', 2);  
plot(hdg_simulator, 'g--', 'LineWidth', 2);
```

## Appendix E Noise Approximation of PW2036 engine

```
% -----  
% MATLAB Script  
% Noise approximation of PW2036-like engine using piecewise function  
% Created by Manuel Amaro (m.a.amarocarmona@cranfield.ac.uk)  
% Cranfield University | 2015  
% -----  
  
clear all;  
Dist = [200 400 630 1000 2000 4000];  
Dist1 = [200 400 630 1000 2000 4000 6300 10000 16000 25000];  
Noise = [90.2 86.3 83.3 79.9 74.2 67.2];  
Noise1 = [90.2 86.3 83.3 79.9 74.2 67.2 61.9 55.2 47.2 34.0];  
plot(Dist1, Noise1);  
  
x1 = [1:Dist(length(Dist))];  
x2 = [Dist(length(Dist)):Dist1(length(Dist1))];  
x = [1:25000];  
  
% Computing first part  
A_count = 0;  
B_count = 0;  
  
m = 3;  
  
for i = 1:m  
    far = length(Dist)-i+1;  
    near = i;  
    B1 = (Noise(near)*log(Dist(far)) -  
Noise(far)*log(Dist(near)))/(log(Dist(far)) - log(Dist(near)));  
    A1 = (Noise(near) - B1)/log(Dist(near));  
    A_count = A_count + A1;  
    B_count = B_count + B1;  
end  
  
A = A_count / m;  
B = B_count / m;  
  
slope = (Noise1(10) - Noise1(6)) / (Dist1(10) - Dist1(6));  
  
hold on;  
plot(x1, A*log(x1)+B, 'r');  
  
% Computing second part  
slope = (Noise1(10) - Noise1(6)) / (Dist1(10) - Dist1(6));  
plot(x2, (slope*(x2 - 4000) + 67.2), 'g');
```

## Appendix F      Functional Requirements

Sl. No.	Requirement Description
1	The generator should predict a path to be flown in relation to the vertical and horizontal dimensions.
2	The path should comply with the maximum limits of aircraft.
3	The path should be optimized according to different profiles in order to reduce fuel consumption, flight time and noise levels.
4	The predicted trajectory requires to comply with the altitude/FL constraints of the flight plan.
5	The predicted trajectory requires to comply with the speed constraints of the flight plan.
6	The guidance system requires to comply with the RNP parameters of the flight plan.
7	The trajectory synthetizer requires to include an estimation of the maximum and minimum time of arrival to each waypoint in order to meet RTA requirements of the Flight Plan.
8	The trajectory must be represented graphically by points separated by distance/time frames along the flight plan route.
9	Maximum response time for full flight plan prediction including flight legs and waypoints shall be reasonable. This requirement applies to flight plans of a reasonable size and complexity.
10	All the data should be referenced according to the Earth Model WGS-86.
11	All altitude data should be referenced to the MSL geoid and standard atmospheric pressure model.
12	The system should include a parameter that defines the Path Definition Error (PDE).